Theses and Dissertations   1. Thesis and Dissertation Collection, all items

1996-09

Decision support for network connectivity planning

Margraf, Jeffrey A.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/32265

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

19970109 024

THESIS

DECISION SUPPORT FOR NETWORK CONNECTIVITY PLANNING

by

Jeffrey A. Margraf

September 1996

Thesis Advisor: Suresh Sridhar
Associate Advisor: Hemant K. Bhargava

Approved for public release; distribution is unlimited.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1996	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE DECISION SUPPORT FOR NETWORK CONNECTIVITY PLANNING	5. FUNDING NUMBERS
6. AUTHOR(S) Margraf, Jeffrey A.	

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT

The aim of this thesis is to design and implement a highly-graphical, computer-based decision support system (DSS) to assist in the design of "optimum" network connectivity plans. The *Web Spinner* DSS is a "proof-of-concept" system which highlights how the marriage of basic decision methodologies with a modern computing environment can be used to create a robust decision support tool.

The basic concepts of decision support systems and their practical value to today's information worker are discussed. The challenge in designing the best network plan is presented along with several examples illustrating the complexities and scale of the problem. The *Web Spinner* DSS is presented as a potential solution to at least part of the network design problem. The capabilities and design principles of the *Web Spinner* are provided along with a tutorial and a sample problem. Finally, some suggestions for improving the *Web Spinner* DSS are reviewed. It is shown that some of these improvements can greatly enhance the value of the *Web Spinner* in supporting decisions related to network connectivity.

14. SUBJECT TERMS Networks, Network Design, Network Planning, Decision Support System			15. NUMBER OF PAGES 138
			16. PRICE CODE

17. SECURITY CLASSIFI- CATION OF REPORT Unclassified	18. SECURITY CLASSIFI- CATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFI CATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

DECISION SUPPORT
FOR
NETWORK CONNECTIVITY PLANNING

Jeffrey A. Margraf
Lieutenant, United States Navy
B.S., United States Naval Academy, 1988

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
September 1996

Author: _____
Jeffrey A. Margraf

Approved by: _____
Suresh Sridhar, Thesis Advisor
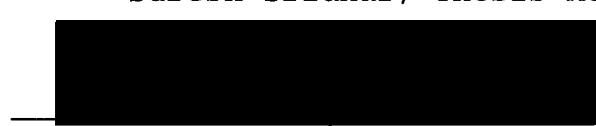
_____
Hemant Bhargava, Associate Advisor
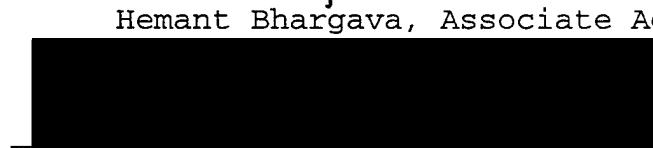
_____
Reuben Harris, Chairman
Department of Systems Management

ABSTRACT

The aim of this thesis is to design and implement a highly-graphical, computer-based decision support system (DSS) to assist in the design of "optimum" network connectivity plans. The *Web Spinner* DSS is a "proof-of-concept" system which highlights how the marriage of basic decision methodologies with a modern computing environment can be used to create a robust decision support tool.

The basic concepts of decision support systems and their practical value to today's information worker are discussed. The challenge in designing the best network plan is presented along with several examples illustrating the complexities and scale of the problem. The *Web Spinner* DSS is presented as a potential solution to at least part of the network design problem. The capabilities and design principles of the *Web Spinner* are provided along with a tutorial and a sample problem. Finally, some suggestions for improving the *Web Spinner* DSS are reviewed. It is shown that some of these improvements can greatly enhance the value of the *Web Spinner* in supporting decisions related to network connectivity.

TABLE OF CONTENTS

ACKNOWLEDGMENT

x

I. INTRODUCTION

A. BACKGROUND

The virtual explosion of computer processing and software capabilities has made the computer as familiar as the telephone and copy machine in the modern office environment. Almost everyone who might need to make an educated decision in any major subject area has access to a computer. The marriage of decision support methodologies and modern information resources has produced some exciting opportunities to make the computer a true office assistant as foretold in many early generation science-fiction movies.

The purpose of this thesis is to design and implement a computer-based, visual decision support tool for designing "optimum" network connectivity plans using several fundamental decision support methodologies. Most similar systems available today are text-based and require the user to mentally picture or manually draw out the resulting network design with each change or iteration. Since networks are inherently visual, an earnest attempt is made to incorporate graphical displays and mechanisms whenever possible to aid understanding and mental comprehension of subtle changes to the model. An emphasis is also placed on ease of use, intuitive methods, and ergonomic design.

The "proof of concept " Decision Support System (DSS) developed as the central focus of this thesis is identified by the title of *Web Spinner*. It was created to assist a network designer in creating "optimum" network connectivity plans. A tutorial for *Web Spinner* is provided as Appendix B and a sample problem in Appendix C. A complete listing of the Delphi (Object-Pascal) code is included in Appendix D.

B. WHY DECISION SUPPORT?

Making the best use of all available information to produce an optimum solution for any problem can sometimes be very difficult. This is especially true in the Information Age as the nature of many problems are becoming less structured and more technical in nature. In addition, the time between problem identification and required solution action is generally decreasing which leaves less time for problem analysis. It may also be a very challenging task to make the best decision when the data or equipment specifications are not directly comparable. This may necessitate a prolonged and involved process to determine value trade-offs between competing proposals to ensure that the "best value" decision is ultimately achieved. The expanding pace of technological change has made some of yesterday's relatively simple decisions into a doctoral thesis that raises more questions than it answers.

There are generally four reasons why an optimum decision may not be made by the decision-maker:

1. The decision-maker possesses incomplete expertise in the subject area.
2. There is too much information for the decision-maker to assimilate and gain a reasonable comprehension of the system.
3. The time between problem identification and required solution action leaves very little time for analysis.
4. The nature of the subject makes it too difficult or computationally intensive to test all possible solutions.

If any of these situations arise, the human decision-maker will still make the best decision he/she can, but it will rarely be optimum. The resulting decision will likely be inadequate, require revision, and be distorted by human mental models and biases. As the decision process becomes less ideal, objectivity and defensibility of the resulting decisions will likely decline.

C. DECISION SUPPORT SYSTEMS

Systems can be designed to assist the decision-maker in confronting some of these problems that lead to non-optimum decisions. These systems are known as decision support systems (DSS). A DSS can be as simple as a sheet of paper and a pencil (to sketch out your problem), a calculator, or a mechanical model of your decision world. The key is to provide a mechanism which assists the individual in arriving at *their* optimum solution. A good DSS also provides analysis and "gaming" algorithms to allow the decision-maker to test different solutions and learn about fundamental relationships imbedded in the problem.

Modern computing resources are increasingly being utilized to develop decision support tools that are very fast, contain knowledge relationships for particular subject areas, and employ graphical tools to display and aid user understanding when large amounts of data are involved. These computerized systems are known as automated decision support systems.

An automated DSS, when properly applied, can be used to assist the decision-maker in over-coming the four identified conditions that may contribute to making non-optimum decisions. The very nature of modern computing systems facilitates the rapid execution of the decision-making process in a more objective and duplicatable manner. This gives the decision-maker the data and material necessary to defend any decision he/she makes and the ability to archive it for future reference. An example of such a system is the central focus of this thesis.

D. SCOPE OF RESEARCH

This thesis and the *Web Spinner* DSS center on the problem of designing optimum network connectivity plans. Designing a network can be a very complex problem due to the remarkably large number of possible network configurations even for a relatively small number of network nodes. Some fundamental decision support methodologies are reviewed and adapted for this purpose.

The author intends to demonstrate that several basic decision models can be combined to create an exceptionally powerful design tool. The *Web Spinner* DSS illustrates how this can be done using a modern programming environment. *Web Spinner* was designed with an interactive graphics interface due to the inherently graphical nature of networks.

E. THESIS ORGANIZATION

Chapter I presents a general discussion of some of the obstacles encountered in the modern decision-making environment. It also presents the application of automated decision support tools to aid the modern decision-maker in overcoming some of these impediments to optimum decision-making. Finally, it introduces an automated DSS which was produced as the central focus of this thesis research.

Chapter II discusses some of the complexities encountered when designing a modern, dedicated-link network. Several examples are provided to illustrate the magnitude of the number of configurations available to connect a network. In addition, it shows how the size of the solution set dramatically increases as the number of nodes in the network increases.

Chapter III presents the *Web Spinner* decision support system. It begins with a general system description and overview of system capabilities. System logic used to design the application is presented along with an explanation of the decision models used in the solution formula. Finally, some information on how the application was actually implemented is provided including primary data structures, design of the *Web Spinner* screens, and a general processing diagram.

Chapter IV discusses some of the limitations present in the *Web Spinner* due to its conception as a "proof of concept" decision support tool. A number of additional features are recommended for inclusion in an improved version of the program. These are provided to assist the reader in looking beyond this particular decision support tool and envisioning how valuable a more complete system could be.

Finally, Chapter V concludes with a retrospective on some of the ideas presented in this thesis and some of the lessons learned during the process. Finally, a discussion on the decision world of the future and how systems of tomorrow will be built upon the foundation blocks of today is presented.

II. NETWORK DESIGN

A. NETWORK CONNECTIVITY PLANNING -- TANGIBLE ISSUES

Modern information networks are becoming larger and more complex as a greater number of systems (nodes) are added to networks each year. This is especially true for networks that bridge several buildings, metropolitan areas, states, or even countries. It becomes no longer possible to run a single cable between all nodes in your network and connect them in a simple daisy chain or bus configuration. Connections between nodes must be leased from a growing number of common carriers (communication vendors) offering a wide variety of link speeds (bandwidths), error rates, pricing structures, and other value-added enmities.

The designer of a modern wide area network (WAN) is faced with determining the best way to connect an increasing number of network nodes in an affordable, reliable, and practical manner. The number of possible configurations dramatically rises as the number of nodes on the network increases. Figure 2.1 shows that there are four potential solutions for a three node network. Figure 2.2 shows that the potential solution set for a simple, four node network dramatically increases with the addition of a single node. In fact, the size of the solution set will grow at an even faster rate as more nodes are added to the network. A five node network has over 750 possible solutions and a six node network has over 20,000.

In a three node network, there are a total of 4 ways to design a network connecting all nodes with full-duplex links (redundant paths permitted):

> 1 three link network (mesh structure)
> 3 three link networks (tree structure)
> **4 *total options***
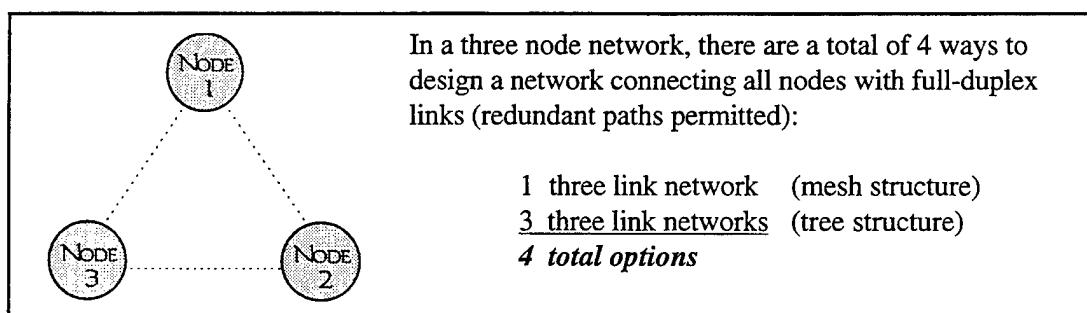
Figure 2.1 A Three Node Network.

In a four node network, there are a total of 40 ways to design a network connecting all nodes with full-duplex links (redundant paths permitted):

1	six link network	(mesh structure)
6	five link networks	(hybrid mesh)
15	four link networks	
18	three link networks	(tree structure)
40	**total options**	
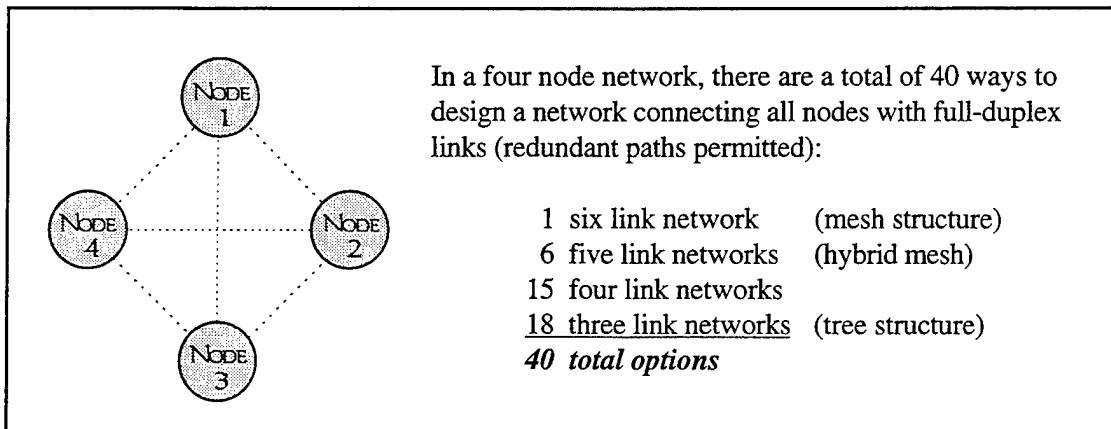
Figure 2.2 A Four Node Network.

If the possible solution set becomes this large for a relatively simple five node network, imagine how large it can get when you attempt to design connectivity plans for a typical 25-50 node (or larger) modern business network. At this point, an inexperienced network designer may begin to feel the pressures of some or all of the four conditions that may lead to non-optimum decisions. In this situation, an automated DSS intended to assist with such a problem could prove invaluable to the novice network designer.

The above discussion has only involved the relatively straight-forward, geometric design of possible network configurations. The problem literally explodes when you consider that there may be ten or more vendors offering many different bandwidth links each with multiple variable attributes. In addition, common carriers are adding an increasing number of options such as line conditioning, connection guarantees, etc., that vary with each vendor considered.

To further appreciate the difficulty of deciding how best to connect your network nodes, let us combine the elements of the preceding paragraphs. Using the three node network, 10 competing vendors, and considering just eight possible bandwidth options, the number of possible network configurations for just three nodes expands to 320:

4 configurations $\times$ 10 vendors $\times$ 8 bandwidths = *320 potential solutions*

If we use the same vendor and bandwidth options to connect our four node network, the potential solution set expands to 3,200:

40 configurations $\times$ 10 vendors $\times$ 8 bandwidth = *3,200 potential solutions*

Using similar math, a five node network has over 60,000 solutions and a six node network grows to over 1.6 million.

Unless the decision-maker is accustomed to designing systems involving such a large potential solution set, the task may seem a bit overwhelming at first. A seasoned network designer will draw upon past experience to eliminate many potential solutions as impractical or infeasible and focus on designs which he/she has successfully encountered in the past. An appropriate DSS can assist the inexperienced decision-maker in designing a better network plan than he/she may contrive alone. However, even a skilled network designer can benefit from an automated DSS which helps him/her quantify their mental weighting mechanisms and choose the optimum configuration from a number of closely competing designs.

B. NETWORK CONNECTIVITY PLANNING --INTANGIBLE ISSUES

So far the network connectivity planning discussion has only included the relatively sterile, physical attributes that outline the problem. The actual task may include a host of intangible (difficult to quantify), political, and "real world" conditions not defined in a structured, textbook assignment. However, this knowledge must be incorporated into any network design to be even relatively successful.

A relevant DSS must allow the user to include these intangible constraints into their network design. A flexible interface that permits the user to add/remove these constraints one-by-one or enmasse and identify how system performance changes may prove invaluable in deciding to accept one network design over another. It should also provide data necessary for the designer to defend any decision against principle managers imposing the constraints upon the designer.

Almost all decision support tools offer a way to archive known facts and resulting decision data and provide "hard-copy," paper reports for future reference. This is a by-product of the process and usually requires no extra work by the user. However, this archived information can provide valuable insight into why a particular plan was accepted if it is questioned at a later date. The resulting design should be reproducible, defensible, archived for future reference, and incorporate a minimum of individual biases.

III. *WEB SPINNER* DSS

A. *WEB SPINNER* OVERVIEW

Web Spinner is a computer-based decision support system designed to assist in the formulation of a network connectivity plan. It permits the user to graphically represent a proposed network of nodes on the screen, input decision criteria information, enter the respective criteria weighting, and graphically define their respective utility curves with regard to the proposed network. *Web Spinner* then designs and displays the optimum network tree structure based on the information provided by the user. Tools are provided for the user to iteratively add and remove criteria from the decision model and *Web Spinner* instantly redesigns the proposed network upon each modification. This allows the user to promptly see various network options and identify how each link characteristic impacts the final network design. Finally, the user is provided with an analysis of over-all network performance and a link-by-link estimated activity table.

The user is also permitted to override the displayed network recommendation by forcing specific links to exist or not exist in the network. This may be required to:

1. Acknowledge political and "real world" factors not defined in the system.
2. Incorporate intangible variables not easily quantified.
3. Incorporate already established links into your design.
4. Identify how system performance changes when incremental constraints are imposed upon it.

This simple function can help the user "learn" more about the proposed network design and "play" with different options. As constraints (links) are applied and removed, *Web Spinner* redesigns the network to optimize the remaining discretionary design decisions. The user is permitted to focus on the problem while the computer makes all the necessary computations to incorporate various combinations of link data into the final design.

11

[Note: The reader may find it useful to become mare familiar with the Web Spinner DSS and its capabilities before reading further. Refer to Appendix A for minimum system requirements and loading the program onto your computer system. Appendix B is a Web Spinner tutorial and Appendix C provides a sample problem. Both Appendixes B and C may be read by themselves or while following along with the application.]

B. SYSTEM DESIGN

1. Graphics and Visual Displays

An earnest attempt was made to use graphical and visually-based screens whenever possible during the design of the over-all *Web Spinner* presentation. Many users feel less intimidated by the complexities of a computer system when using simple and seemingly basic tools to input and manipulate data.

Well-designed graphical tools also tend to be more intuitive to the user and allow for the representation of large amounts of data in an easy to understand, uncluttered format. The graphical illustration of data may further allow the user to see trends that may be hidden in reams of raw data. In addition, sometimes it may be a faster and more efficient process for the user to convey information to the computer through a graphical tool than requiring the user to manually input every data point.

Hardware and software advances over the past decade have made the almost unrestricted use of graphical tools possible and easier to implement. *Web Spinner* takes advantage of some of these advances and graphically depicts the proposed network and how it changes as decision criteria are modified. Extensive use was made of standard Microsoft Windows® components to give it a familiar look and feel to most users.

2. General Decision Models

Web Spinner uses two primary decision models when designing the proposed network. The first is the equating of standard criteria data to a common utility value and

12

applying user-defined weights and utility scales. This provides a mechanism to directly compare criteria data that are not directly comparable in their original form. All decisions are then made based upon the combined utility values of each option vice the actual data values. The second decision model used by *Web Spinner* is the application of the minimum/maximum spanning tree model common to the introductory chapters of most Operations Analysis or Graph Theory books.

a. Utility-Based Design

The *Web Spinner* DSS uses three criteria to identify link attributes: Set-Up Cost; Monthly Cost; and Bit Error Rate (BER). (These three criteria were selected for the initial version of *Web Spinner*, but other link attributes could have been used and should be incorporated in a production quality system.) The user is expected to collect this data for each full-duplex link between every node pair. This data is then input by the user into data matrices provided on the *Web Spinner* Inputs screen.

The user must identify the relative importance of each criteria with respect to each other in the design of the network. This is done in the form of weighting assignments. The user assigns a percentage value (or relative weighting) to each of the three criteria for a total of 100%. A possible assignment may be: Set-Up Cost = 50%; Monthly Cost = 30%; and BER = 20%. This assignment indicates that Set-Up cost is the most important criteria to the user and is as important as the other two criteria combined. Monthly Cost is less important, but should be considered above BER when designing the network. *Web Spinner* uses these values when computing the relative "goodness" of each link option.

The user must also identify their respective utility curves for each of the criteria. These utility curves are independent of each other and instruct *Web Spinner* as to

the relative importance of different values of each criteria. Two example utility curves are shown in Figure 3.1.

Set-Up Cost Utility

BER Utility
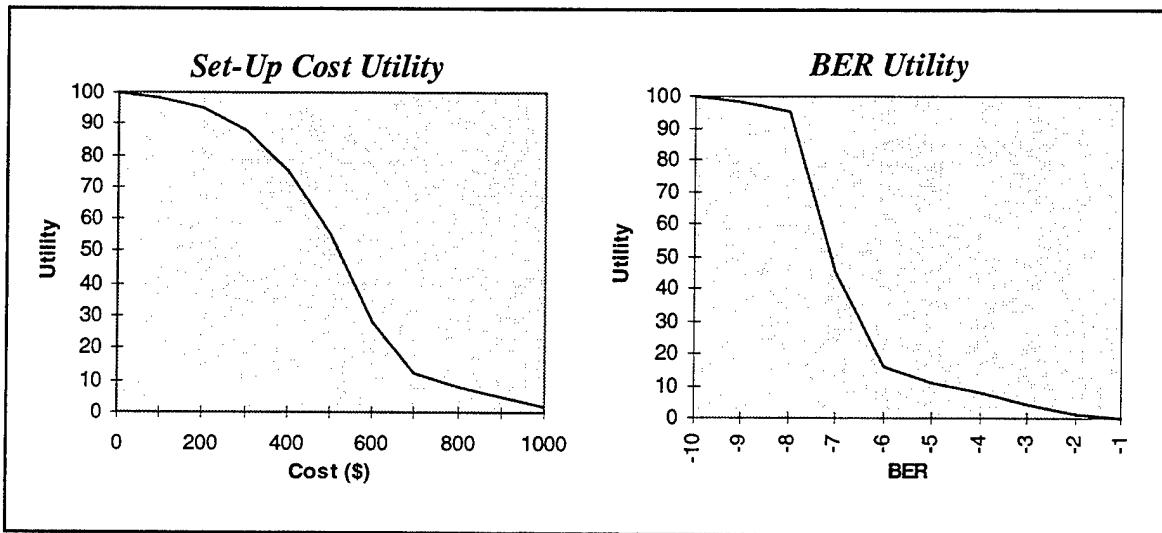
Figure 3.1 Example Utility Curves.

These two graphs indicate that the user is generally more concerned with maintaining a low BER than controlling Set-Up Cost for the proposed network design. The curve associated with Set-Up Cost begins with a gradual downward slope and does not decrease as fast from left to right as the BER curve. The user feels that the "goodness" (or utility) of increasing criteria values does not decrease as fast for Set-Up Cost as it does for BER. The user is willing to tolerate a higher link Set-Up Cost before he/she is willing to tolerate a higher BER value.

To obtain the utility value of a specific criteria point, locate the criteria point on the X-axis and draw a vertical line until it intersects the data curve. Draw a horizontal line from this intersection to the Y-axis. The point of the Y-axis intersection is the respective utility value or worth for the specific criteria point. Figure 3.2 shows that the utility value of a $350 cost to establish a link is approximately 82. As the cost

increases, the respective utility value will decrease according to the defined utility trend curve.
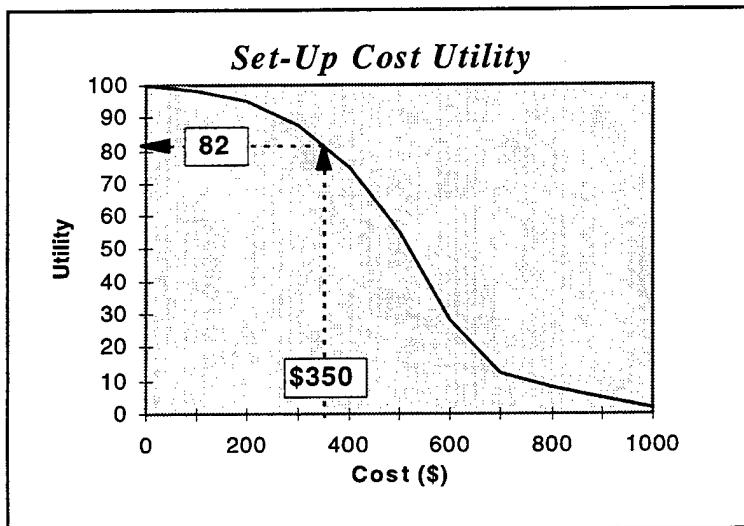
Figure 3.2 Extracting a Specific Utility Value.

Utility curves may not seem initially intuitive to those individuals not familiar with decision support tools. However, utility curves are an important technique in conveying user valuation of changing criteria values to many modern decision support systems. After the user has practiced creating utility trend curves and adopted them into their thought process, their value will soon become apparent.

Once criteria weights and utility curves have been defined, *Web Spinner* uses the following formula to determine the total utility or "goodness" value of each link:

Link Utility = (Set-Up Cost Weight × Set-Up Cost Utility Value) +
                 (Monthly Cost Weight × Monthly Cost Utility Value) +
                 (BER Weight   × BER Utility Value)

Web Spinner first computes the total utility value for each potential link in the network. These unitless values are then used with the minimum/maximum spanning tree models to design the recommended network.

b. Minimum/Maximum Spanning Tree Model

The minimum spanning tree model is a "greedy algorithm" used to create the "optimal" tree structure in a spatially-defined network. The minimum spanning tree is a tree structure that connects all nodes in a network so that the total branch (link) lengths are minimized. The resulting network is considered "optimal" in that it connects all the nodes in a network at a minimum total distance or link units. The solution method is simple and easy to follow.

The first step is to select any node in the network as your starting point and draw a link between it and its nearest neighbor. This creates a spanning tree consisting of two nodes. (It does not matter which node you begin with as the result will always be the same.) The next step is to select the closest node not presently in the spanning tree. If there is a tie for the closest node, arbitrarily select one of them. Draw a line between this node and the nearest node in the growing tree structure. The process is repeated until all nodes are included in the spanning tree. The result is referred to as the minimum spanning tree. An example of this technique is shown in Figures 3.3 and 3.4.

In a five node network, there are $(N^2 - N)/2$ possible full-duplex links that may exist in the network. In this case, $(5^2 - 5)/2 = 10$ links. These potential links are represented by dashed lines. The length of each link is represented by the attached number.
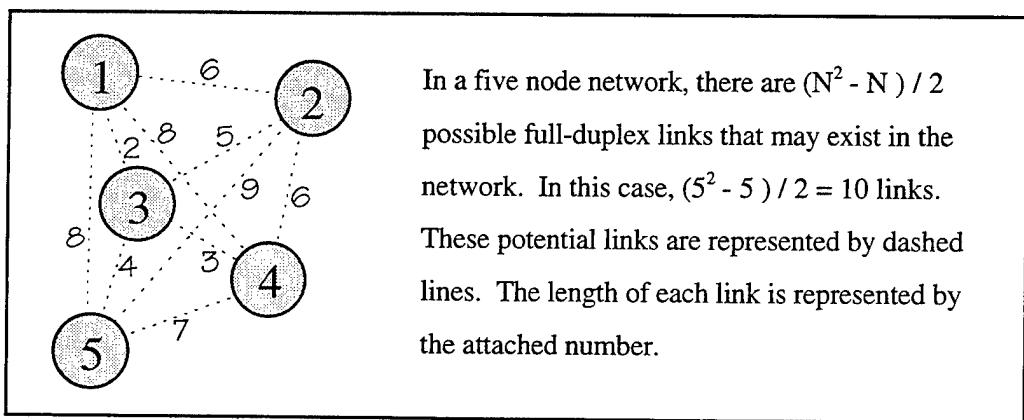
Figure 3.3 A Five Node Network.

16
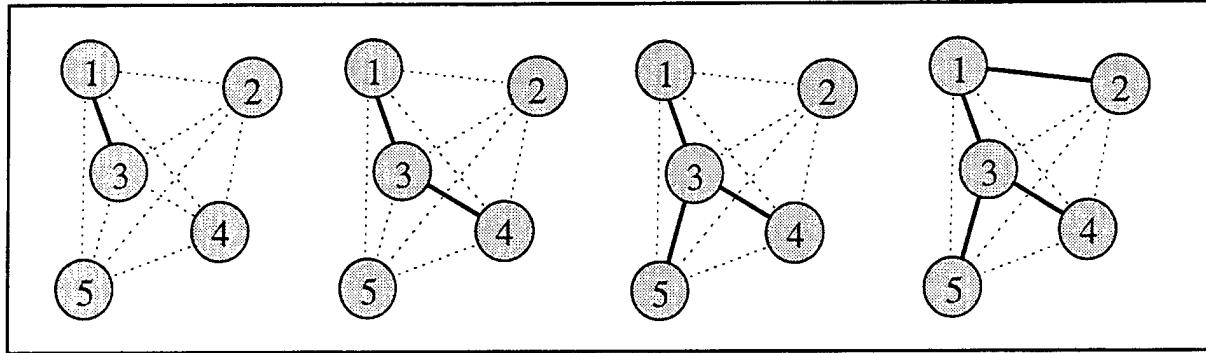
Figure 3.4 Solving the Minimum Spanning Tree.

This process can also be used to create a maximum spanning tree. The only difference is that you search for the node farthest away from your spanning tree structure and select it. The above five node network is solved using this technique and shown in Figure 3.5.
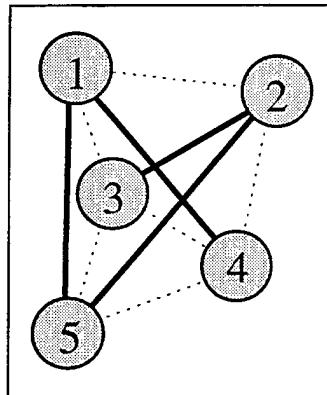
Figure 3.5 Maximum Spanning Tree.

3. Designing the Recommended Network

Web Spinner uses the above decision methodologies to design the recommended network. When the user wants to design the network based on link lengths, *Web Spinner* uses the minimum spanning tree method and the screen distance between nodes to design

the network. This technique typically creates an aesthetically pleasing tree structure that makes sense to the network designer.

However, when the user designs the network using the assigned decision criteria, weights, and utility scales, *Web Spinner* uses computed link utility values and the maximum spanning tree model. This process can sometimes produce a graphical tree structure with overlapping links and a form that seems illogical. Nevertheless, *Web Spinner* correctly designs the optimum tree structure based on the combined utility value of the input data. Many times the associated link data differs from the relative link lengths and our mental model of what a tree structure should *look* like may be challenged by the result. An example of what appears to be an illogical tree structure was shown in Figure 3.5. By definition, the network shown is a tree structure but it does not seem to *look* quite right to the mental eye of most designers.

Web Spinner continually designs and draws the recommended network configuration based on the current set of user-defined data. The relatively fast speed of the computer processor is used to do all of the required computations. In fact, it is so fast that it appears instantaneous to the user. Each time a single data point, criteria weight, utility value, or node position is modified by the user, *Web Spinner* recalculates each value and redraws the proposed network.

4. "User Learning"

An excellent DSS will also provide mechanisms to facilitate user learning about the fundamental relationships imbedded within their decision world. *Web Spinner* attempts to do this by allowing the user to decide which combination of decision criteria they wish the program to use to design their network. A checkbox mechanism is provided which allows the user to choose one, any two, or all three criteria. The user can quickly and efficiently check and uncheck the criteria checkboxes and *Web Spinner* immediately recalculates new

18

link utility values and redraws the network based on the selected criteria only. This allows the user to gain insight into how each criteria affects the proposed design.

In addition, *Web Spinner* allows the user to require that particular links exist in the network. The application then redraws the network and recalculates all performance data by optimizing the remaining network options. This can be used by the user to immediately see how constraints imposed on the system affect the design and the resulting reduction in network performance.

5. Network Performance Analysis

Web Spinner generates a network performance analysis report every time a new network is designed. The information presented in this analysis includes:

1. Total network set-up cost.
2. Monthly network cost.
3. A *Physical Link* table including average traffic loading and mean packet delay for each physical link.
4. A *Logical Link* table including the physical route and expected route delay for each logical connection in the network.
5. Expected network delay.

Total network set-up cost and monthly network cost are self-explanatory. However, the remaining data requires a bit more analysis and calculations to generate. The following discussion will identify intermediate variables and show the logic used to arrive at this information. A brief description the network analysis techniques used by Web Spinner is presented below. [Ref. 1 & 2]

Network traffic is measured in packets per second. Full-duplex links are used and a shorthand symbol, O/D, is used to refer to each origin/destination pair or link. The variables used in this analysis are:

1. N = Total number of links in the network
2. C = Uniform link capacity (in bits per second)
3. i = Link identification number

4. γ = Total network traffic = $\Sigma_{\text{all O/Ds}}$ (Traffic)
5. $1/\mu$ = Packet size (bits per packet)
6. *λ_i* = Average number of packets which travel on link *i* per second
7. ρ_i = $\lambda_i / (\mu C)$ = Traffic intensity on link *i*

A physical link is a link that *physically* exists in the proposed network. (i.e. you can visually see it drawn in the network design) The physical link data is the information provided for each physical link in the proposed network. To calculate the mean delay for an average packet, we must first determine the routes that make up each logical link.

A logical link refers to each notional connection in the network. A logical link exists for each node pair combination. The route is determined by selecting the single path required to traverse the network between the two nodes and recording all the nodes encountered along the way.

After all logical routes are determined, the amount of traffic (packets) required to travel between each node pair is added together by physical links. This way, we determine how many packets per second will be expected to traverse each physical link. This value is denoted by the variable λ_i.

The mean physical link delay in seconds, *$E\{T_i\}$*, is then calculated using the following formula:

$$E\{T_i\} = \frac{1}{Capacity(pkts\,/\,\sec) - Average(pkts\,/\,\sec)}$$

$$E\{T_i\} = \frac{1}{(\mu_i C - \lambda_i)} \quad \text{or} \quad \frac{\rho_i}{\lambda_i(1 - \rho_i)}$$

The logical link delay, or route delay, is then calculated by adding up all the mean physical link delays which form each route.

Finally, the expected network delay is calculated using the following formula:

20

$$T = \frac{\sum_{i=1}^{N} \lambda_i \times E\{T_i\}}{\gamma}$$

Web Spinner computes all of these calculations each time the proposed network or the associated decision criteria are modified. Figure 3.6 shows our five node minimum spanning tree network and some required performance criteria. Figures 3.7, 3.8, and 3.9 show the resulting analysis data for this network.

| _Traffic Requirements_ | |
Link	Traffic
1-2	5
1-3	7
1-4	4
1-5	6
2-3	8
2-4	5
2-5	7
3-4	10
3-5	4
4-5	12

Link Capacities = 9600 BPS

$1/\mu$ = 128 bits per packet

Total (γ) = 68 pkts/sec
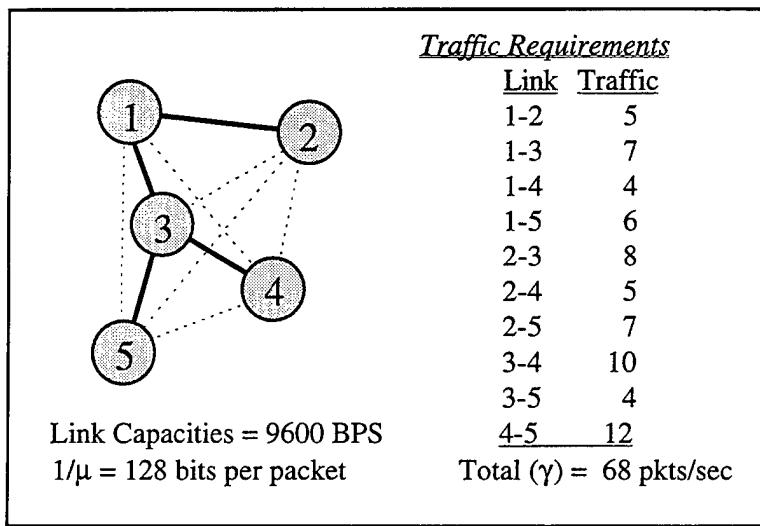
Figure 3.6 Five Node Network and Performance Requirements.

Physical Links

Link	Avg pkts/sec	$E\{T_i\}$ (delay)
1-2	25 pkts (5+8+5+7)	0.02 sec (1/[9600/128 - 25])
1-3	37 pkts	0.0263 sec
3-4	31 pkts	0.0227 sec
3-5	29 pkts	0.0217 sec

Figure 3.7 _Physical_ Link Data.

Logical Links		
Link	Route	Exp. Route Delay
1-2	1-2	0.02 sec
1-3	1-3	0.0263 sec
1-4	1-3-4	0.049 sec (0.0263 + 0.0227)
1-5	1-3-5	0.0481 sec
2-3	2-1-3	0.0463 sec
2-4	2-1-3-4	0.069 sec
2-5	2-1-3-5	0.0681 sec
3-4	3-4	0.0227 sec
3-5	3-5	0.0217 sec
4-5	4-3-5	0.0445 sec

Figure 3.8 Logical Link Data.

$$T = \frac{\sum_{i=1}^{N} \lambda_i \times E\{T_i\}}{\gamma}$$

$$T = \frac{25(0.02) \ + \ 37(0.0263) + \ 31(0.0227) \ + \ 29(0.0217)}{68\,pkts\,/\,sec}$$

$$T = 0.0413\sec onds$$

Figure 3.9 Expected Average Network Delay.

C. PROGRAM IMPLEMENTATION LOGIC

1. *Web Spinner* Menu Structure

The user interface for *Web Spinner* is divided into five distinct modules:

1. *Web Spinner* Desktop
2. Data Input
3. Criteria Weights
4. Utility Curve Definition
5. Performance Analysis

All five of these modules are represented by individual screens and can be accessed by tabs displayed along the bottom of each screen. Figure 3.10 shows the *Web Spinner* graphical desktop and the five tabs present on the bottom of all module screens.

The desktop is a graphical panel which acts as the drawing canvas for the user to depict the relative locations of the prospective network nodes. It also is the surface upon which *Web Spinner* creates and displays proposed network links. Figure 3.10 shows a sample network or nodes and the links drawn by the *Web Spinner* application. Figure 3.11 shows examples of the remaining four module screens.
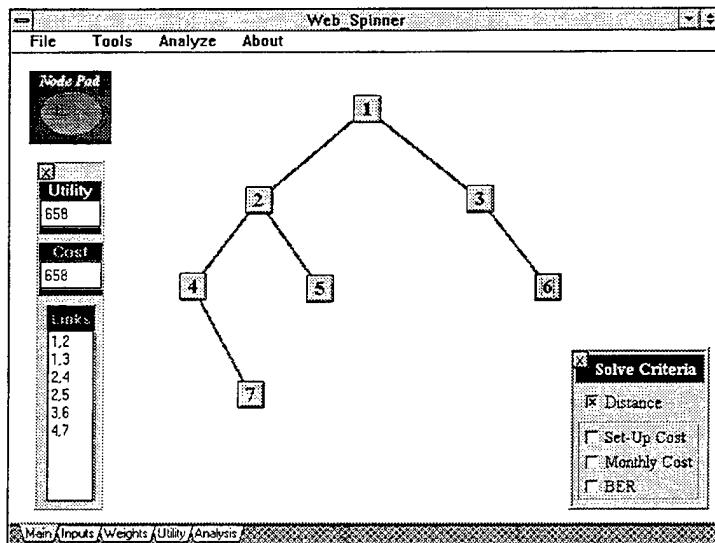
Figure 3.10. Web Spinner Desktop and Module Tabs.

The data input module (Link Characteristics) is a screen tool that allows the user to enter traffic requirements data along with set-up cost, monthly cost, and BER data for each possible link in the network. It also displays computed distance and combined utility values for each potential network link in a read-only format.

The weights and utilities modules provide simple tools that allow the user to graphically enter relative criteria weighting and utility curves. The analysis module presents the user with an analysis of the current network configuration using the logic presented in the previous section.
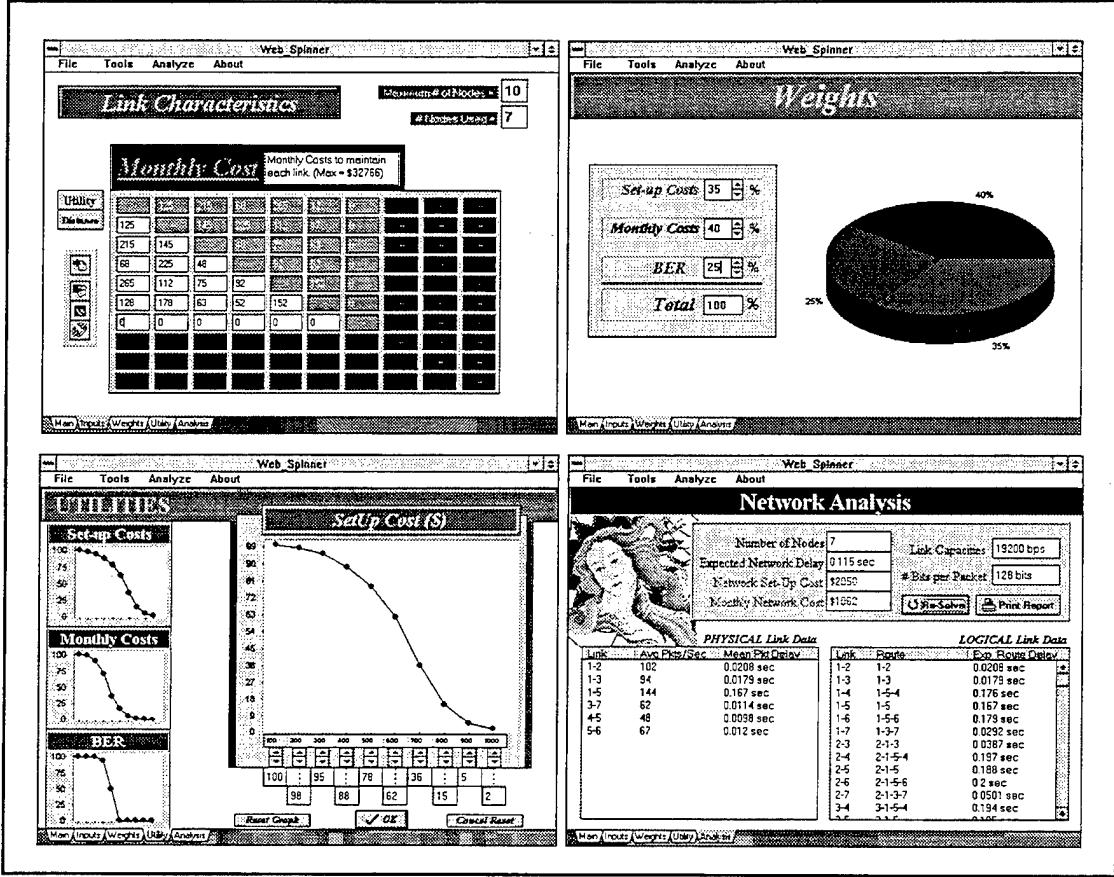
Figure 3.11 Data Input, Weight, Utility, and Analysis Modules.

2. Primary Data Structures

A matrix is an ideal data structure for maintaining data on each possible link on the network. Each cell in the matrix can be used to represent a specific link and store specific data about that link. For example, a 5×5 matrix can be used to represent a five node network. The cell represented by row three and column two references the potential link between nodes two and three. This is shown graphically in Figure 3.12.

If full-duplex links are used to design your network, then you only need half a matrix to represent the links. This is because the link represented by cell 3,2 is the same as

Figure 3.12 Matrix Cell Representing the Link
Between Nodes 3 and 2 (Cell 3,2).

the link represented by cell 2,3. In addition, cells referenced by the same column and row

number are unnecessary as it would be illogical to establish a link beginning and ending at

the same node. Figure 3.13 represents the same 5×5 matrix with "half" of the matrix

blacked out to represent the type of matrix used by the *Web Spinner* to store data.

Figure 3.13. "Half Matrix" Showing Link 2,3
(same as Link 3,2).

This "half matrix" structure is the format used on the data input screen. The user

is expected to fill in the lower left half of the matrix and *Web Spinner* simultaneously fills

25

in the upper right half to complete the matrix. This effectively halves the time required by the user to input data into the *Web Spinner* matrices.

Web Spinner uses nine matrices to store and maintain data about the network. Four matrices are integer matrices that contain the Traffic Requirements, Set-Up Cost, Monthly Cost, and BER rating for each potential network link. Two additional integer matrices, Distance and Utility, are used to hold the computed distance between nodes (length) and total utility value of each link. A real number matrix is used to store the computed packet delay for each physical link. A character-based matrix is used to record the routes that constitute each logical link between every node pair. Finally, a specialized matrix is used to store the status of potential links as existing, non-existing, or locked in the recommended network. This matrix is used by the network drawing routine to graphically depict the proposed network on the *Web Spinner* desktop.

These matrices are continuously manipulated by the user and the *Web Spinner* application during the design of the network. A number of smaller arrays and variables are used throughout the program, but offer little value-added information for the reader.

3. General Web Spinner Decision Process

Figure 3.14 is a graphical illustration of the *Web Spinner* decision process. The set-up cost, monthly cost, BER, and traffic requirements matrices are filled in by the user. The entire process is executed when modifications are made to the network or the input data. Each time the process is executed, the remaining five matrices are recomputed and included in the solution process. The relatively quick speed of the computer makes this complex process appear instantaneous.
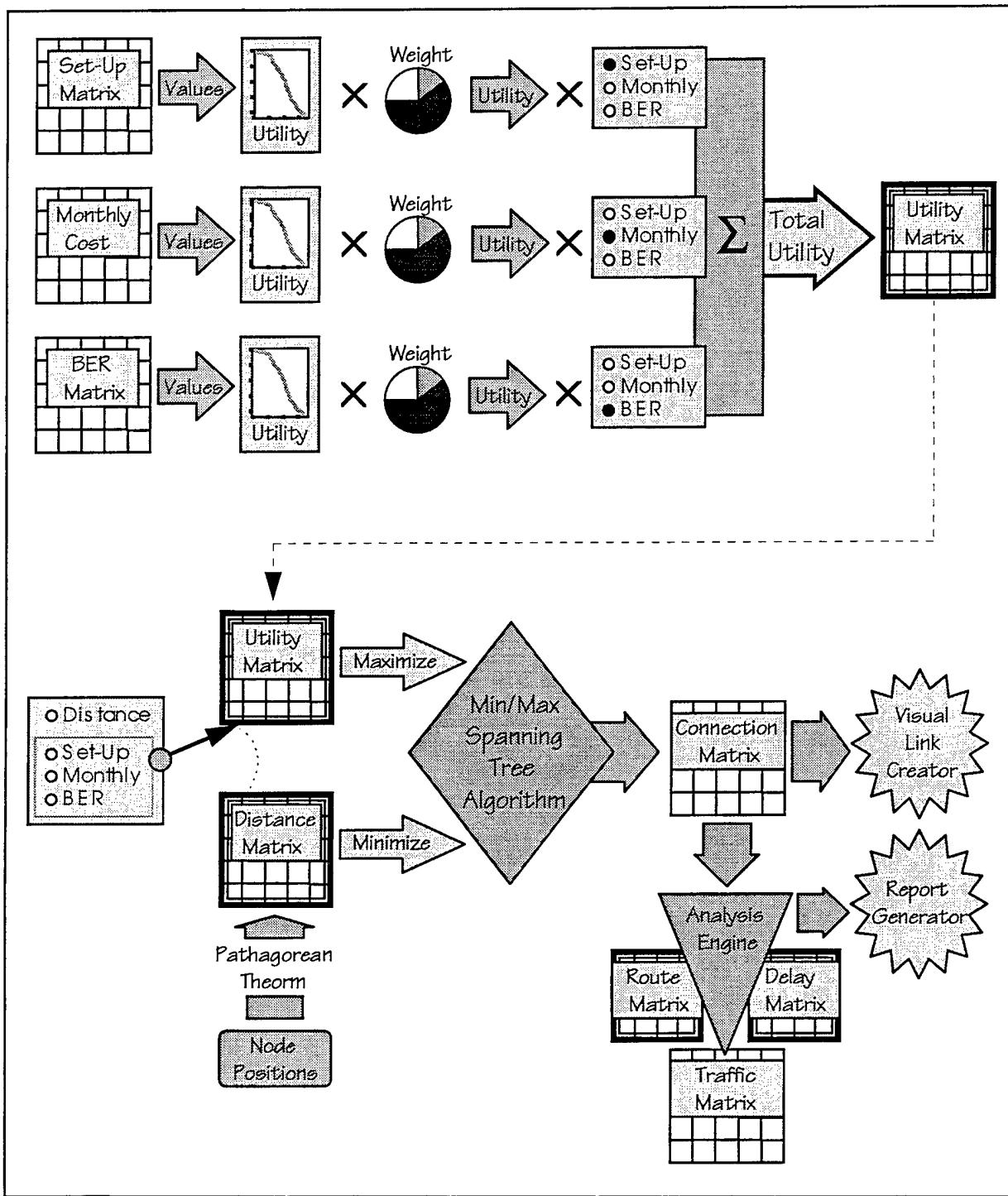
Figure 3.14 *Web Spinner* General Process Diagram.

27

IV. WEB SPINNER LIMITATIONS & RECOMMENDATIONS

A. PROGRAM LIMITATIONS

Web Spinner was developed as a "proof of concept" decision support system to show the usefulness of such a network design tool. As a result of its prototype origin, there are some limitations in the application including:

1. A single link speed (Uniform Link Bandwidth) for all network connections is not practical for a "real world" problem.
2. Only one set of link data can be stored in the *Web Spinner* data matrices at a time.
3. Ten network node limitation.
4. Only three criteria are used which might not accurately depict the decision world of prospective users.
5. The decision algorithm always creates an "optimum" tree structure. Additional configuration options should be available.
6. *Web Spinner* can only be used on machines capable of running Microsoft Windows applications.

Most of these deficiencies would be corrected in a production quality version of the system. In addition, a number of features should be added to make *Web Spinner* easier to use and a more complete decision support tool for the user. These additional features are covered in the next section (Section B, Recommendations).

1. Uniform Link Bandwidths

A true optimum network would utilize the best link bandwidth for each connection. Otherwise, some links will have a large percentage of unused bandwidth while others will be approaching their upper limit. If bandwidths are optimized at the link level instead of the network level, the total cost of establishing and operating a network should be considerably less. This network-level, uniform link bandwidth constraint decreases network throughput efficiency, produces uneven link loading rates, and generally creates a more expensive network.

29

2. Single Set of Link Data

The *Web Spinner* matrices only provide for the input of data for a single potential link between each node pair. This was done for two reasons. The first is that the uniform link bandwidth constraint limits the amount of data required to that of a single link between each origin and destination node. However, when the user changes the uniform link bandwidth for the network, he/she must then input new data appropriate for the new link speed.

The second reason is caused by the choice of implementation platform. Windows 3.xx and the Borland Delphi 1.0 compiler enforce a 64 Kilobyte (KB) restriction on the amount of RAM (Random Access Memory) that can be used by the program stack and data segment of each application. (This restriction is advertised to have been corrected with Delphi 2.0 running with the Windows-95 operating system. See Appendix D for a description of the Delphi programming language and a listing of the program code.)

Web Spinner uses nine matrices (discussed in Chapter III, Section C) plus a number of arrays to store and manipulate network data. As the number of nodes increases, the number of cells in each matrix increases with the square of the number of nodes (i.e. 2 nodes creates a $2^2 = 4$ cell matrix and 3 nodes require a $3^2 = 9$ cell matrix). Nine growing matrices can quickly consume a lot of computer memory. In addition, as the number of nodes increase, the size of the internal memory stack must grow to accommodate the recursive routines used to determine logical routes and network performance data. The highly graphical nature of the *Web Spinner* screens also consumes a large amount of memory as graphics are stored in a memory-intensive bitmap format (*.bmp). Therefore, memory size requirements for the application dictated that only a single set of matrices could be maintained for each link at one time.

30

3. Ten Node Limitation

The size of each network is limited to ten nodes on account of the 64 KB memory restriction presented in the previous section. This limitation was not discovered until the analysis module was developed for *Web Spinner*. The analysis module uses two recursive procedures which dynamically expand and reduce the program memory stack as required to complete the routine. When more than ten nodes are used, the memory stack can exceed the available memory space causing an "Out of Memory" error to occur. Windows 3.xx then terminates the *Web Spinner* application.

4. Three Criteria

Three criteria were chosen for the initial version of *Web Spinner*. These criteria are: Set-Up Cost; Monthly Cost; and Bit Error Rate (BER). They were chosen because they are considered to be common to the decision world of most system designers when designing a dedicated link network. However, it is probable that these are not the only quantifiable criteria that would be considered in the design process. A mechanism should be added which allows the user to define their own criteria and accompanying data in order to make *Web Spinner* a more useful and complete decision support tool.

5. Exclusively Tree Structures

The minimum spanning tree is the combination of links that connects all network nodes with a minimum total length (or other appropriate unit). Likewise, the maximum spanning tree is the tree structure which maximizes the total length using the minimum number of links. For most dedicated link network designs, the Minimum/Maximum Spanning Tree models creates the ideal network solution. However, if redundant routes are required between links, spanning tree models can no longer be used as dual routes between any two nodes violates the definition of a tree.

31

Additional configuration modeling techniques would be useful to a network designer. This would allow him/her a greater freedom of choice and more control of the process when determining *their* optimum network design. Whatever the reason, the minimum/maximum spanning tree structure may not always be optimum to the network designer.

6. Windows® Environment Only

Web Spinner was developed on a IBM-compatible computer system running Microsoft Windows for Workgroups 3.11. Standard Windows® components were used to enhance graphic effectiveness and give it a common look and feel to most users. As a result, *Web Spinner* requires a similar system to run effectively. The current implementation is not available to Apple Macintosh or UNIX system users.

B. RECOMMENDATIONS

It is recommended that the identified limitations be corrected in a production-quality version of the system. In addition, a number of new and expanded features are recommended for inclusion:

1. Include network performance analysis into the design.
2. Allow specification of several additional network criteria such as minimum network response time, maximum link loading rates, and logical link error rates.
3. Allow for file storage and retrieval.
4. Allow for import of link data directly from the service provider.

This added functionality could dramatically increase the effectiveness of *Web Spinner* as a decision support tool.

1. Network Performance Analysis

The network analysis module was added to provide post-design information to the user. As a result, the computed network analysis information is not included in the

decision process. It would be very useful to the decision-maker if this data were incorporated into the solution formula along with a sensitivity analysis showing how final performance data will change as input data is modified.

2. Additional Criteria Specification

In addition to the provided network analysis information, a seasoned network designer may wish to specify that the resulting design meet several criteria specifications. Examples include specifying minimum network response time, maximum link loading rates, and logical link error rates. Pre-specification of final network performance characteristics is likely to be required by a "real-world" network designer and should be included in the decision process.

3. File Operations

Data input consumes a large amount of the users time during a *Web Spinner* session. In addition, when the user terminates his/her session, all input data is lost. It would be very useful for the network designer if he/she could input network data once and save it for later sessions. File storage of this data would be very useful if the network was frequently changing or new information becomes available at a later date. The addition of a relational database could provide a convenient way to store and retrieve this data.

The only way the user can currently retain any session information is through a paper copy of the network analysis report or individual screen printouts. This may be adequate for archival purposes, but it does not allow the decision-maker to use the system to graphically show why this decision was the optimum solution for the conditions present at the time the decision was made.

4. Import of Vendor Data

Another very useful feature would allow *Web Spinner* to import link data directly from the vendor. This could be done by loading data from a floppy disk or directly from the vendor's database via a computer modem. This could dramatically cut down on the amount of time consumed by data input and allow for the latest vendor information to be incorporated into your design.

A number of problems would have to be overcome to implement this function including data incompatibility, differing vendor data, and data acquisition (not all vendors may have this information available electronically). However, a local relational database management system could be used to assist in the capture and retention of this data.

V. CONCLUSION

A. RETROSPECTIVE

The *Web Spinner* is not a state-of-the-art decision support tool, nor is it a system that could be used to completely solve a real-world problem. *Web Spinner* is, however, an excellent proof-of-concept system which underscores the usefulness of such tools both today and in the future. *Web Spinner* succeeds in providing an effective graphical environment and presentation appropriate for today's sophisticated information-worker. Simple decision models and graphical displays are combined to create a powerful, albeit limited, environment for designing network connectivity plans.

The knowledge foundation gained by implementation of the *Web Spinner* DSS can be improved by incorporating many ideas contained within this thesis. Many lessons were learned along the way and the author hopes that they can be used to not only solve the current design issue at hand, but incorporated into the growing knowledge base for computer-aided solving of real world problems.

B. MORE TO COME

Decision Support Systems will play an increasing role in the aid of human decision-making in the future. They are not meant to replace the human decision-maker, but assist them in a synergistic way. The computer can be harnessed to take care of all the time-consuming and structured calculations of the decision while allowing the user to concentrate on the fundamentals of the decision. The electronic computer excels at pre-programmed, calculation-intensive problems -- something that can be time consuming and tedious to the human computer. The human mind is better at pattern recognition and incorporating non-quantifiable, complex insights into the decision process.

With the expansion of the business world to a global scale and the increasing decision complexities accompanying the emerging Information Age, many problems are becoming more technical and data intensive than every before. Many of today's decisions have far-reaching implications and recognize a diminishing tolerance for error. Most errors and fundamental design flaws can be caught long before they are implemented through the use of advanced decision support technologies and methods. Tomorrow's systems will be more robust and build upon the lessons learned today.

<u>APPENDIX A</u>. GETTING STARTED WITH *WEB SPINNER*

A short guide is provided here to assist the user in loading and configuring their system to use the *Web Spinner* application.

A. System Requirements

Web Spinner was developed on an Intel 80486DX2-66-based computer system running Microsoft Windows for Workgroups 3.11. The following minimum system is required to properly run the *Web Spinner* application:

1. Intel 80386DX processor with Math Co-Processor (80387)
2. Windows 3.xx or Windows-95 Operating System
3. 4 MB RAM (8 MB recommended)
4. 64 KB available conventional memory
5. 1 MB available hard disk space
6. 256 color VGA monitor with video card
7. Mouse
8. Printer (required for "hardcopy" reports)

B. The *Web Spinner* Program Disk

The *Web Spinner* program disk contains three files which must be properly loaded onto your system to execute the program. These three files are:

1. **Webspnr.exe** - The *Web Spinner* executable file.

2. **Bivbx.dll** - A Dynamic Link Library (DLL) source file which provides extended Microsoft Windows objects and methods used in the *Web Spinner* application. It assists the *Web Spinner* in drawing pictures and graphical components on the screen during program execution. *Web Spinner* will not load without this file present.

3. **Chart2fx.vbx** - This is a Visual Basic Component (VBX) file which provides the visual methods and functionality for graphical charts in the Microsoft Windows operating environment. *Web Spinner* will not load without this file present.

C. Getting Started

Loading the *Web Spinner* application is as simple as following steps 1-2-3:

1. Step 1 - Using the *File Manager* in Windows 3.xx, the *Explorer* in Windows-95, or simple DOS commands, load the **webspnr.exe** file onto the hard drive. It is recommended that the user create a separate directory on the hard drive and load **webspnr.exe** there.

2. Step 2 - Load the **Bivbx.dll** and **Chart2fx.vbx** files into the C:/windows/system directory on the boot drive. These files must be loaded here as the operating system will be looking for them as the *Web Spinner* program is executed.

3. Step 3. Execute (Run) **webspnr.exe** from the hard drive and *Web Spinner is ready to use*. Figure A.1 depicts the *Web Spinner* initial welcome page. Appendix B provides a tutorial designed to teach the fundamental capabilities of *Web Spinner*. For those who wish to learn more about using the application, Appendix C contains a real world sample problem and a step-by-step process of solving it using the *Web Spinner*.
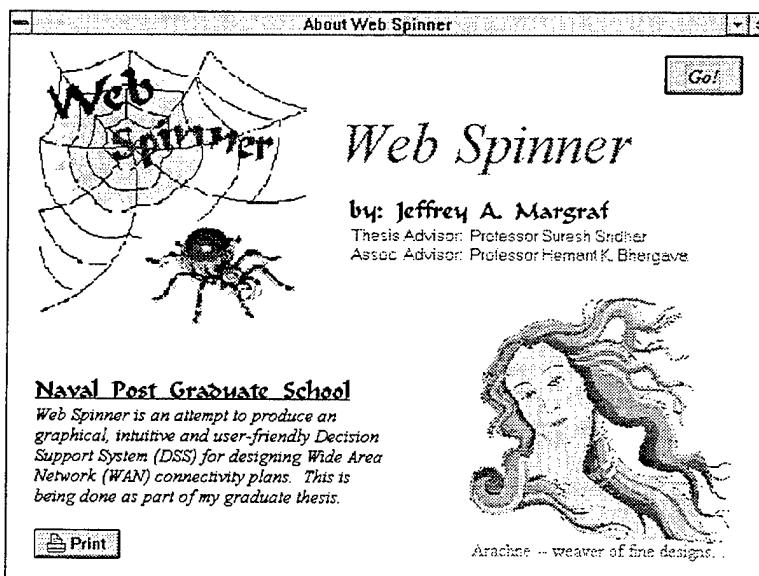
Figure A.1 The Web Spinner Welcome Page.

APPENDIX B. *WEB SPINNER* TUTORIAL

Web Spinner was designed to be a graphical, intuitive, and user-friendly Decision Support System (DSS). Extensive use was made of Microsoft Windows® standard visual components and data manipulation tools. A simple tutorial is provided here to highlight its graphical simplicity and to serve as an instructional mechanism for potential users. *Web Spinner* attempts to assist the user in defining the "optimum" tree-based network based on a number of selectable criteria: Set-Up Cost, Monthly Cost, and Bit Error Rates (BER).

[Note: Unless otherwise noted, when reference is made to "click" on a menu item, the user is expected to position the on-screen pointer over the item and press the left mouse button.]

A. Getting Started

When the program is launched, the user is first greeted by a welcome screen (Figure B.1) which provides some basic information about the program.
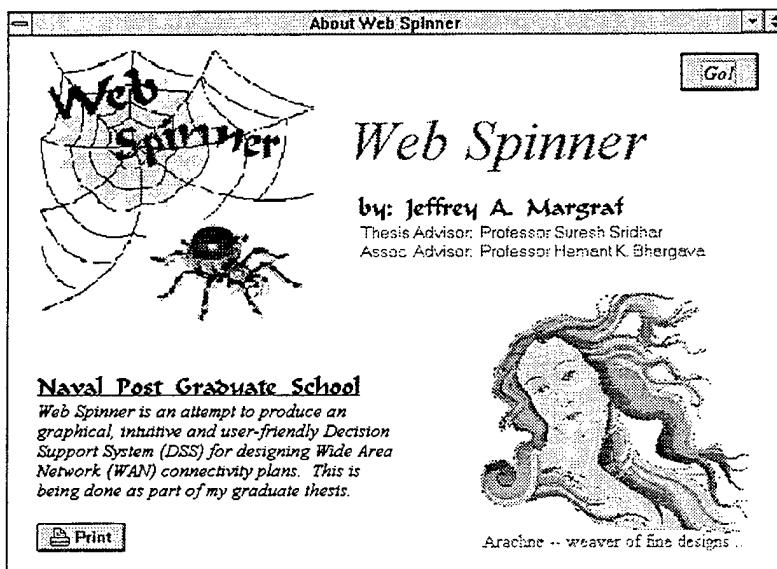
Figure B.1 *Web Spinner* Welcome Page.

The *Web Spinner* logo is depicted in the upper left of the screen and a picture of the Greek maiden Arachne is shown in the bottom right corner of the window. For an explanation of the presence of Arachne throughout the program and the motivation she provides, simply click with the mouse anywhere on her picture.

Click on the *"Go!"* button to proceed.

B. The *Web Spinner* Desktop

The first screen presented to the user is the *Web Spinner* desktop. The desktop is one of the five tabbed modules which make up the *Web Spinner* DSS. The five tabbed sections are:

1. **Main** (The *Web Spinner* Desktop)
2. **Inputs** (Data Input)
3. **Weights** (Criteria Weighting Assignment)
4. **Utility** (Utility Curve Definition)
5. **Analysis** (Network Performance Analysis)

Figure B.2 shows a blank desktop with the five tabbed sections displayed along the bottom of the window. Each section may be chosen by clicking on the appropriate tab with the left mouse button.
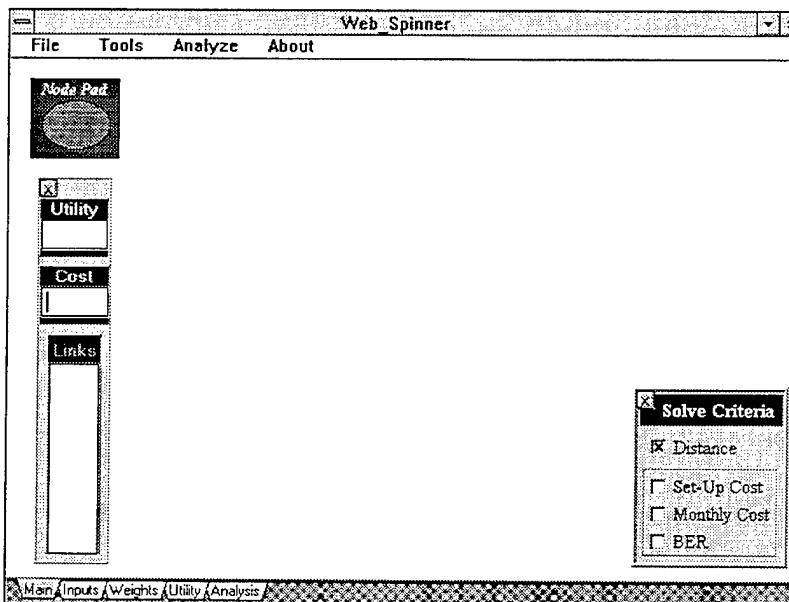
Figure B.2 The *Web Spinner* Desktop.

Initially, the desktop is configured as shown in Figure B.2 with the *Node Pad, Link Info,* and *Solve Criteria* tool boxes present. All visual components can be moved anywhere on the screen by pressing the left mouse button with the pointer over the component and dragging it to a new position. The move process is completed when the mouse button is released. In addition, almost all components can be hidden from view by clicking on the "x" button in the upper left corner of the component. All components may be restored to the desktop by selecting the appropriate menu item from the pull-down menus on the top of each screen.

40

C. Pull-Down Menus

A number of standard pull-down menus are present at the top of most windows. They are used to either invoke a desired procedure (method) or to display hidden tool boxes. A description of each menu item and available short-cuts (shown in parenthesis) are provided here for clarification.

Figure B.3 shows the *File* pull-down menu and the two methods available for selection. The *Print Screen* (CTRL + P) selection sends a copy of the currently displayed screen to the printer. The *Close* selection ends your *Web Spinner* session.

File	Tools	Analyze	About
Print Screen CTRL+P			
Close			
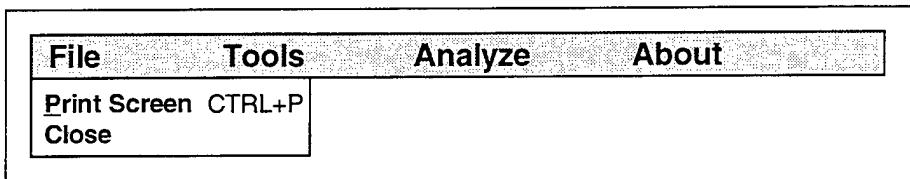
Figure B.3 File Pull-down Menu.

The *Tools* pull-down menu, shown in Figure B.4, is used to hide or unhide two tool boxes that are used with the *Web Spinner* Desktop. The *Link Creator* (CTRL + L) tool is used to force a link to exist between any two nodes in your proposed network. The *Link Info Box* (CTRL + I) displays the links chosen to connect your network for the given criteria. In addition, it also shows the calculated network set-up cost and total utility for the displayed network configuration.

File	Tools	Analyze	About
	Link Creator CTRL+L		
	Link Info Box CTRL+I		
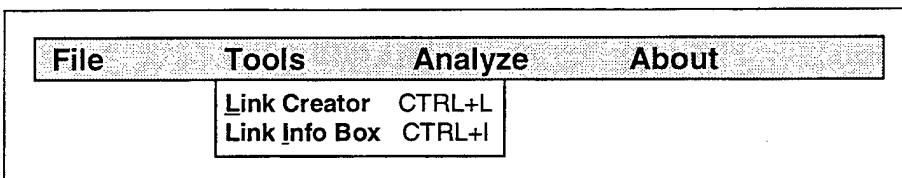
Figure B.4 *Tools* Pull-down Menu.

Figure B.5 depicts the *Analyze* menu. The *Analyze* menu contains two tools which are used to change basic information used by *Web Spinner* to design the proposed network. The *Solve Criteria* (CTRL + C) tool is a checkbox tool which determines which criteria (Distance, Set-Up Cost, Monthly Cost, BER) or combination of criteria should be used to plan the network. As criteria are selected and de-selected, *Web Spinner* automatically redraws the network based on the selected criteria only. This can give the user valuable insight into how each criteria affects the final outcome.

41

File	Tools	Analyze	About
		Solve <u>C</u>riteria CTRL+C	
		Basic <u>D</u>ata CTRL+D	
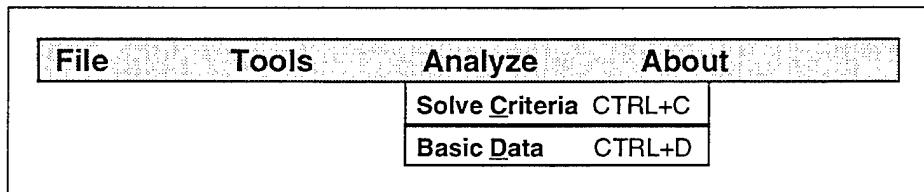
Figure B.5 *Analyze* Pull-down Menu.

The *Basic Data* (CTRL + D) tool box is used to input two basic elements of the network: uniform link bandwidth (link speed in bits/second) and packet size (bits/packet). When either element is modified, *Web Spinner* recomputes all the network information found on the *Analysis* page.

The final pull-down menu is the *About* menu and is shown in Figure B.6. The *About* menu contains two information tools: *About* and the *Link Legend*. The *About* (CTRL + A) selection returns the user to the initial *Web Spinner* screen showing basic information about the program. The *Link Legend* tool is a moveable panel which simply depicts the different colored links that can be displayed in the network and what they represent.

File	Tools	Analyze	About
			<u>A</u>bout WS CTRL+A
			Link Legend
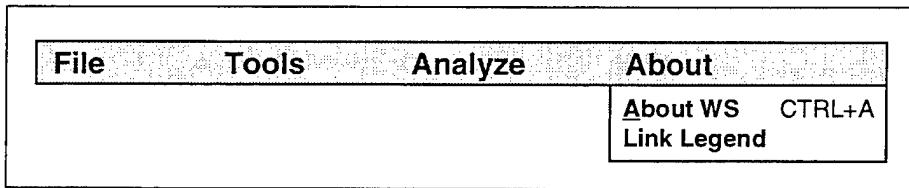
Figure B.6 *About* Pull-down Menu.

D. Creating Your Network

The first step in creating your proposed network is to define the network nodes. Node generation and spatial positioning are done on the *Web Spinner* desktop. Click on the magenta oval on the *Node Pad* to generate a new network node. Once the new node appears on the node pad, the user may drag it to any position on the desktop. The user should position the nodes on the screen so that they roughly approximate the geographic positions of the actual network. A maximum of 10 nodes may be defined in this version of *Web Spinner*.

As each node is positioned on the screen, *Web Spinner* will begin drawing links to show the best network tree structure based on the scaled screen distance between nodes.

These links will also be identified in the *Link Info Box* on the left side of the screen. Figure B.7 shows a sample network of nodes displayed on the *Web Spinner* desktop.
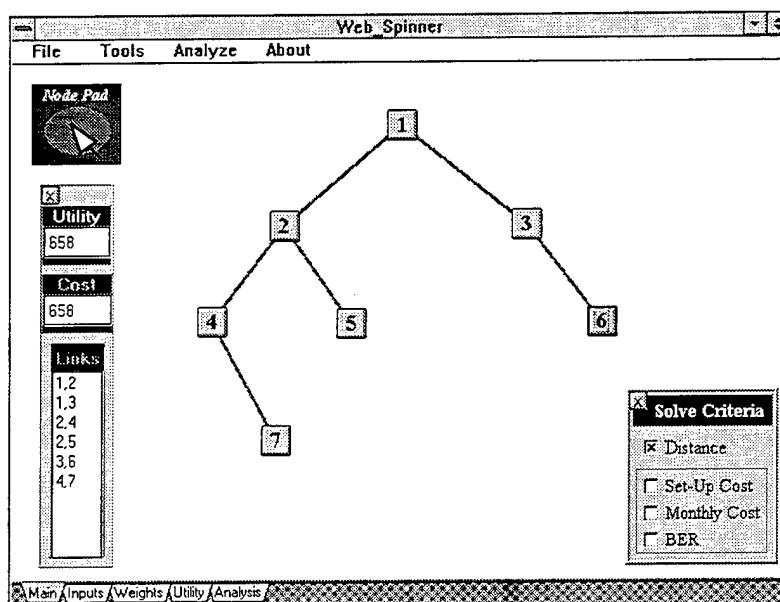
Figure B.7 Defining Network Nodes on the Desktop.

A node may be repositioned at any time during your session by simply dragging it to a new desired position. *Web Spinner* will automatically re-draw the network based on this new screen position and the computed distances between nodes.

If you wish to delete a node, simple click with the right mouse button on a node and *Web Spinner* will ask if you wish to delete the node (Figure B.8). If you delete the node, *Web Spinner* will re-number the nodes to ensure continuity, remove the respective node data from the criteria matrices, and redraw the network. If you regenerate the node at a later time, the respective node data will have to be re-entered into the criteria matrices.
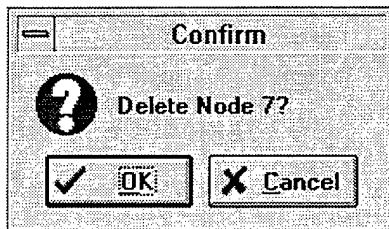
Figure B.8 Deleting a Node.

You may generate a node at any time during your session. New nodes will always be assigned the next number in the numerical sequence between one and ten. Remember, the maximum number of nodes that may exist at any time is ten. A dark cross pattern will appear in the oval on the *Node Pad* when the maximum number of nodes have been defined to indicate the inability to generate additional nodes.

E. Network Data

Once you are satisfied with the number of nodes in the network and their relative positions on the desktop, the next step is to define traffic requirements between network nodes as well as set-up costs, monthly costs, and bit error rates (BER) for potential links. Select the *Inputs* page to enter this data. An example of this screen is shown in Figure B.9.
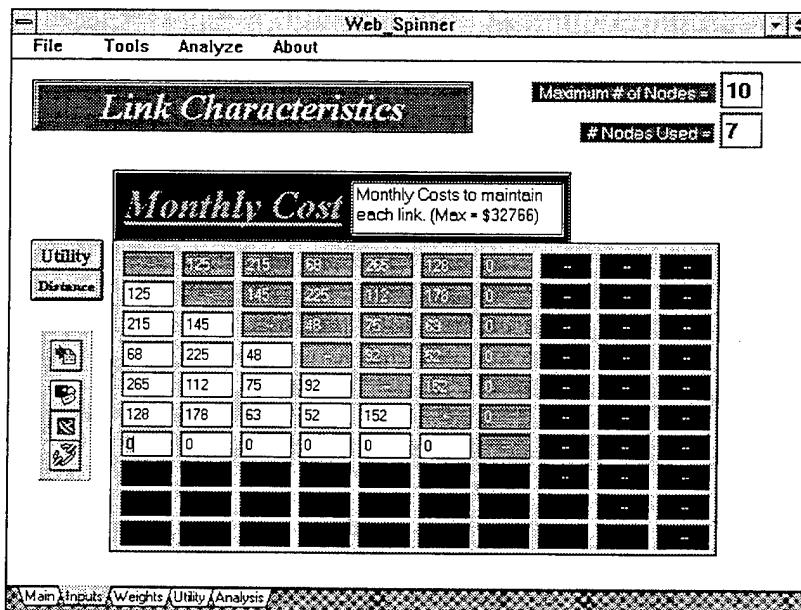
Figure B.9 Defining Network/Link Data.

Web Spinner maintains network and link data in a series of six matrices that may be accessed by the user. These matrices are expanded and contracted throughout your session based on the number of nodes currently defined on the desktop. Each link is represented by the convergence of the representative row and column. For example, the matrix cell represented by row two and column three represents the link between nodes two and three. All links are considered full duplex so only half the matrix needs to be defined -- *Web Spinner* fills in the second half. Therefore, the *Inputs* screen displays a matrix with the top-right half darkened and uneditable.

44

The first two matrices, Utility and Distance, are automatically computed by *Web Spinner* and are available on the *Inputs* screen as read-only information for user reference. These matrices may be displayed by simply clicking on the respective button on the left side of the screen.

The four remaining matrices are: Traffic Requirements; Set-Up Costs; Monthly Costs; and BER. They may be accessed by the speed buttons shown on the left side of the screen. If you position the pointer over a speed button, *Web Spinner* will tell you what matrix the button activates after a short pause (about 1.5 seconds). As each button is pressed, the respective matrix is displayed on the screen along with a brief description of the data *Web Spinner* expects to be input into the matrix.

F. Input of Network Data

In order for *Web Spinner* to compute an accurate report of your complete network, it requires the estimated traffic requirements between nodes. Select the first speed button on the *Inputs* screen. The current traffic values will be displayed. Select the top white matrix box (row 2, column 1) and input the expected amount of packets that need to be transmitted between nodes one and two each second. Once the value is input, press the tab button on the keyboard to move to the next input box. *Web Spinner* will fill in the darkened half of the matrix as you input each value. You may change these values at any time during your session.

Once you are satisfied with the Traffic Requirement matrix values, select the second speed button to activate the Set-Up cost matrix and input these values. Do the same for the Monthly Cost and BER matrices using the input guidance provided.

G. Defining Criteria Weighting

Web Spinner uses three basic criteria to assist the user in defining their "optimum" network: Set-Up Cost; Monthly Cost; and BER. In order for *Web Spinner* to determine the importance of each criteria to the user, he/she must define their relative importance with respect to each other. This is done on the *Weights* page shown in Figure B.10.

The *Weights* screen shows the three criteria and a pie chart which graphically represents their assigned weights. Use the spinbuttons to increase or decrease the assigned relative weighting for each criteria. The pie chart and the total box automatically update as criteria values are modified.
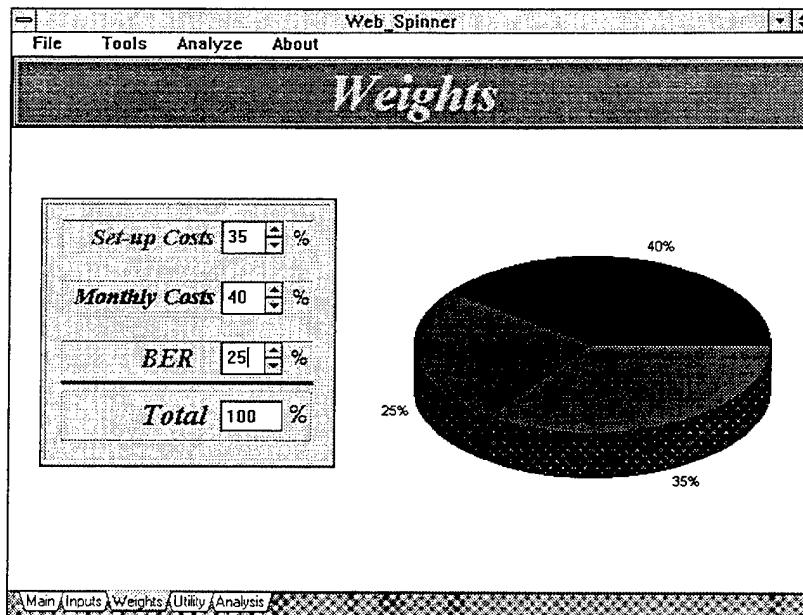
Figure B.10 Criteria Weighting.

H. Defining Criteria Utility Curves

Utility graphs are used to determine the value trade-offs of the user with respect to each of the selection criteria. Select the *Utility* page as shown in Figure B.11. This screen displays three small charts along the left side of the screen identifying the current utility values for each of the three criteria. The larger chart is used to modify these utility charts.
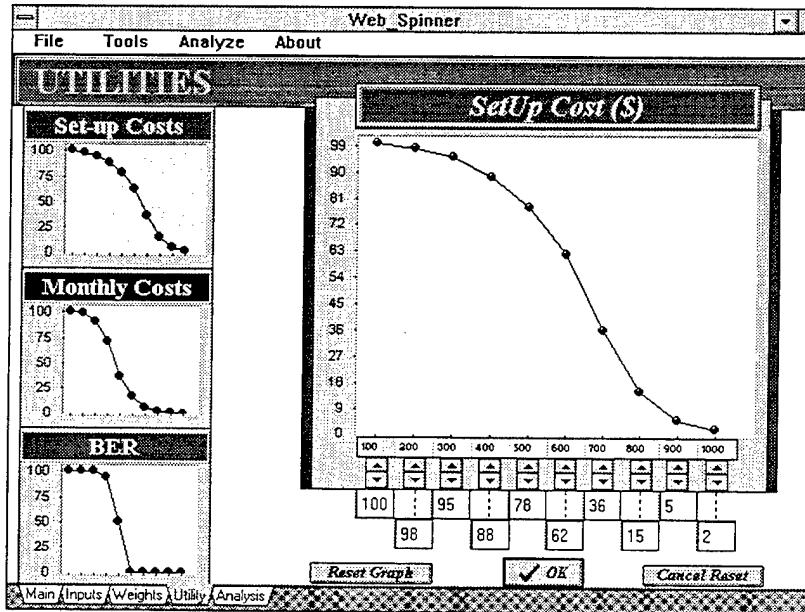
Figure B.11 Defining Utilities.

46

Initially, the Set-Up Cost utility graph is shown in the larger chart. To select either of the other curves, simply double click on the smaller graph and it will appear in the larger, modification chart. Any changes you made to the previous graph will be reflected in the respective, smaller chart.

To change utility values, use the appropriate spinbuttons to increase or decrease the value for each data point. To reset a graph to a "risk-neutral," straight-line curve, select the *Reset Graph* button below the graph. To cancel this reset or any other changes, click on *Cancel Reset* to return the graph back to the initial modification point.

To complete the modification, either select the *OK* button or double click on the next chart to transfer the modification graph curve to the smaller graph.

I. Defining Required Links

Many times specific network links may be required within the network. These reasons may be political, intangible, or to incorporating already established links in the proposed network design. *Web Spinner* allows you to force specific links to exist to reflect these unquantifiable realities. This method also allows the user to override the tree-based structure of the network by defining a series of locked links that form a non-tree structure.

Return to the desktop. There are two ways to force specific links to exist. If the desired link is already shown on the desktop, simply click on the link and *Web Spinner* will ask if you wish to lock the link. (see Figure B.12) If the *OK* button is selected, the link will change color from magenta to black and will always be included in the proposed network solution regardless of its respective utility.
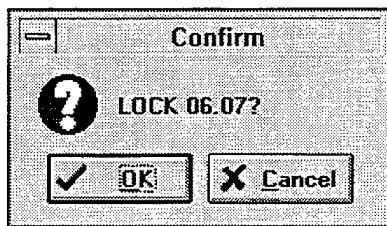
Figure B.12 Locking a Link.

The second way to lock a link in place is used when the desired link is not currently depicted on the screen. Click on the *Tools* pull-down menu and select the *Link Creator* (CTRL + L). A box will appear on the screen which you may input the two nodes that you wish to create a locked link between. (see Figure B.13) The values must represent nodes currently depicted on the screen. Select *OK* to complete the operation.

47
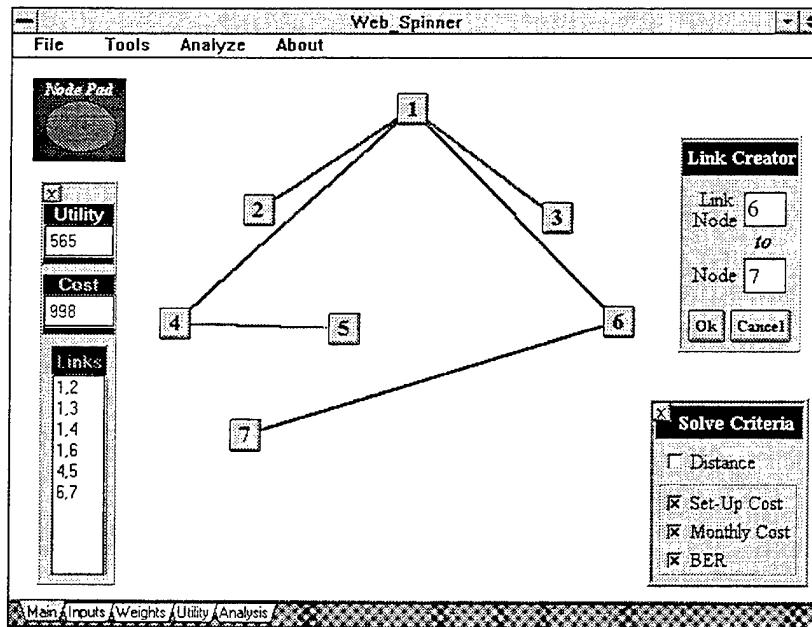
Figure B.13 Using *Link Creator* to Lock a Link.

To unlock a link, simply click on a locked link and the user will be asked if they wish to unlock it. (see Figure B.14) If the *OK* button is selected, the locked nature of the link will disappear and it will return to the magenta color. *Web Spinner* will then treat the proposed link as any other and use the selection criteria to determine the "optimum" network composition.
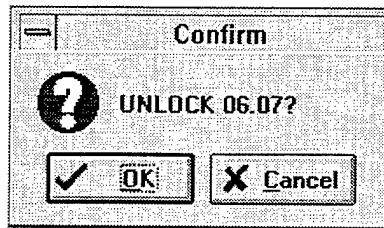
Figure B.14 Unlocking a Link.

J. Analyzing the Proposed Network

Web Spinner will now assist the user in designing the "optimum" network based on the data provided. On the desktop, the *Solve Criteria* box can be used to select which criteria you want *Web Spinner* to use to solve your proposed network design. Check the *Set-Up Cost* box and *Web Spinner* will re-draw the network using set-up cost as the sole selection criteria as shown in Figure B.15.

48
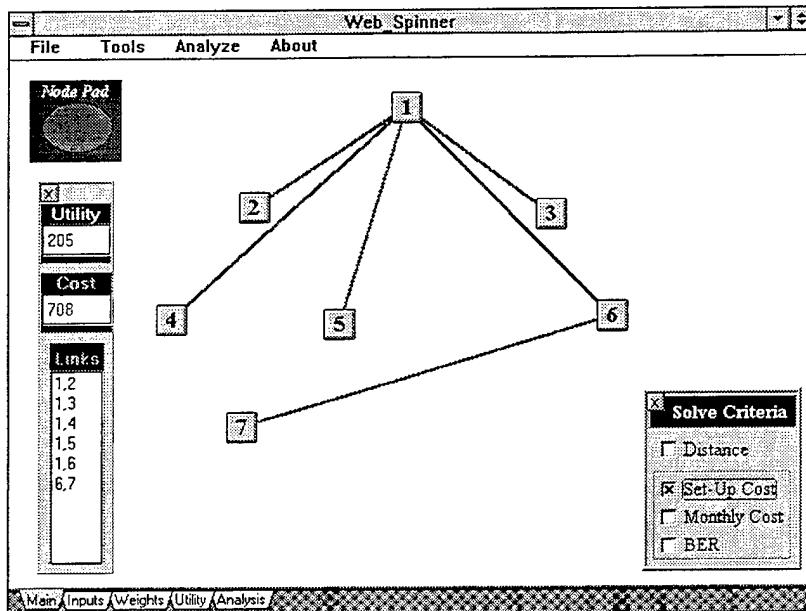
Figure B.15 Network Based on Set-Up Cost Only.

You may select any combination of criteria to design your proposed network. *Web Spinner* re-draws the network each time any modifications are made. The "optimum" tree-based network is generated by computing the respective utility values for each link and selecting the combination of links which create a tree structure with the highest over-all utility value. The following formula is used:

Total Link Utility = [(Set-Up Cost Weight) × (Set-Up Utility) × (Check Value)] +
[(Monthly Cost Weight) × (Monthly Utility) × (Check Value)] +
[(BER Weight) × (BER Utility) × (Check Value)]

The check value is equal to a one if the respective box is checked or a zero if it is not checked. Figure B.16 shows the proposed network with *Monthly Cost* and *BER* criteria selected.

The user can now gain insight into the proposed network behavior as different criteria are selected or as specific links are locked and unlocked. In addition, nodes can be added and removed from the network and link criteria values, weights, and utility curve values can be modified. Each time a modification is made, *Web Spinner* re-draws the network based on the current set of data and criteria selected.

49
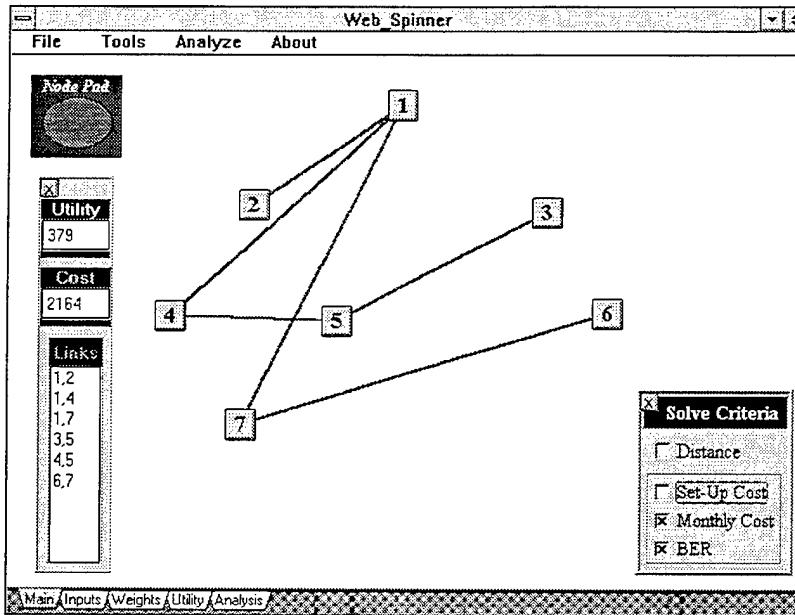
Figure B.16 Proposed Network Using Two Criteria.

K.   Finishing Up -- the Final Network Report

Once you have settled on your final network configuration, *Web Spinner* will generate a network analysis report containing essential network statistics. Select the *Analysis* page.

As the page appears on the screen, the data on the screen is recomputed to ensure all recent network modifications are included. If the uniform link bandwidth selected at the beginning of the session is inadequate for the proposed network, a dialog box will appear with the minimum bandwidth required to implement your network. (see Figure B.17)
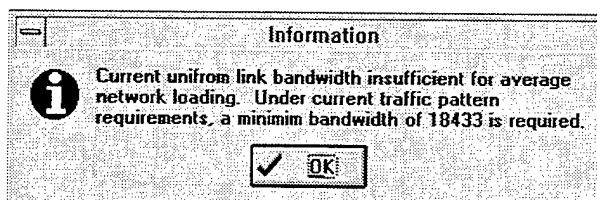
Figure B.17 Warning, Link Speed Insufficient.

50

After the user acknowledges the message, the *Basic Data* box appears to allow the user to input a new uniform link speed. (see Figure B.18) Input the new value and select *OK*.
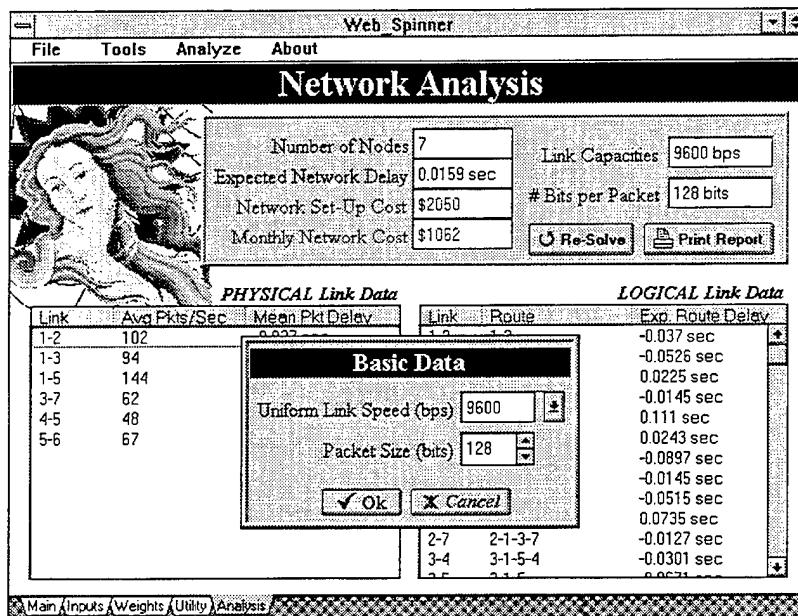
Figure B.18 Changing Link Speed Using Basic Data Box.

To recompute the report analysis based on the new link speed, select the *Re-Solve* button. Figure B.19 shows what a final, satisfactory report may look like.
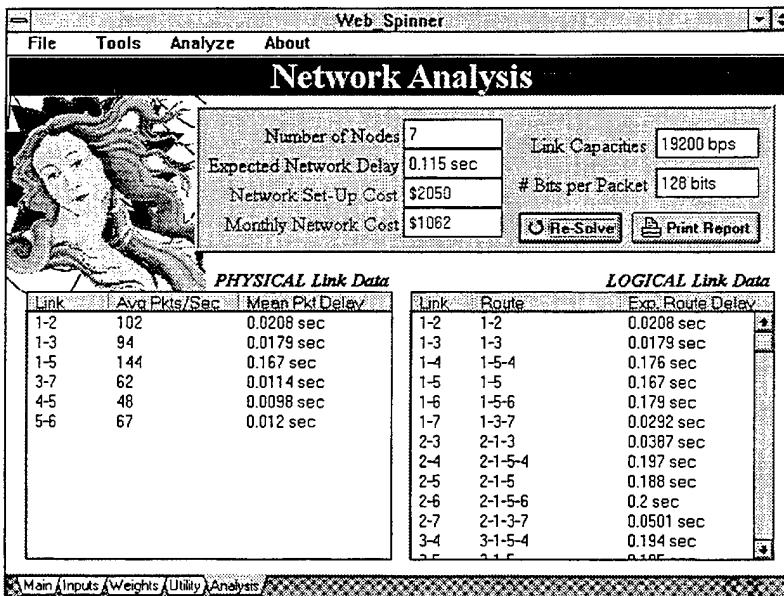
Figure B.19 Final Report.

To obtain a paper copy of this report as well as all of the criteria values used to create the network, select the *Print Report* button.

Each time the user makes modifications to the network, *Web Spinner* recomputes this data which may be referred to at any time during the session. The final printed report can be used to justify and document the network design process for future reference.

A sample problem is presented in Appendix B for those who wish to further explore the capabilities of the *Web Spinner*.

APPENDIX C. SAMPLE PROBLEM

A sample problem is provided here to show how one might use the *Web Spinner* decision support system to solve a real world problem. It is designed so that the reader may either skim this section separately or follow along using the *Web Spinner* program. *Web Spinner* is a "proof of concept" system vice a production quality system, so some minor artificialities do exist within the exercise.

A. The Problem

You are assigned as the J-4 (Logistics) Systems Officer for the Joint U.S. Transportation Command (USTRANSCOM) at Scott AFB near Belleville, Illinois. You have been tasked to design a network connectivity plan for a new system designed to allow USTRANSCOM to coordinate critical logistics among a number of CONUS naval base supply centers during periods of crises or war. In addition, this network will be used by those supply centers to coordinate intra-Navy logistics plans among themselves. The ten nodes for the prospective network are listed below and shown graphically in Figure C.1:

1. U.S. Transportation Command, Belleville, IL
2. National Military Command Center, Washington D.C. (Joint Chiefs of Staff)
3. Naval Base, Groton, CT
4. Naval Base, Norfolk, VA (U.S. Atlantic Fleet)
5. Naval Base, Charleston, SC
6. Naval Base, Mayport, FL
7. Naval Base, Bremerton, WA
8. Naval Base, Oakland, CA
9. Naval Base, San Diego, CA (U.S. Pacific Fleet)
10. Naval Base, Gulfport, MS

The network is to be designed with "wartime" or peak anticipated bandwidth requirements and utilize common communication links for simplicity. High reliability and low error rates are required as it will serve the critical logistics needs of a wartime fleet. The system administrator informs you that the proposed network will utilize a very small packet size of 16 characters per packet. You do some quick math and determine that the proposed network packet size will be 128 bits.

A link already exists between USTRANSCOM and the Command Center in Washington D.C. A logistics link is also in operation between the naval bases in Norfolk and San Diego. Both of these links have approximately 64 kilobits per second (KBPS) unused bandwidth that you have permission to use for the new network.
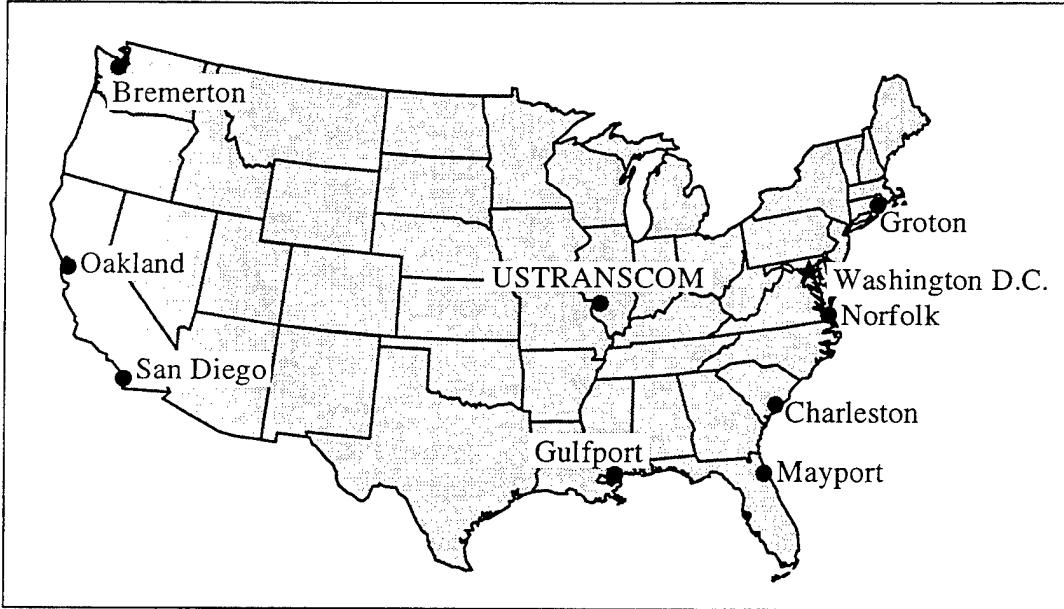
Figure C.1 Prospective Nodes in the Network.

Due to a high commitment rate of Operation and Maintenance (O & M) Funds for this fiscal year, minimal funding is available to establish the network. However, the J-8 Budget Officer believes that additional funds will be approved for future year operations of the network.

B. Using *Web Spinner* to Solve Your Problem

You realize that the proposed network nodes are geographically dispersed so you will have to lease commercial links to provide the required connectivity. Since you have a tight network set-up budget and already have two 64 KBPS lines available, you decide to incorporate these links into your network. You also decide to make 64 KBPS the uniform link speed standard for the initial network.

You launch the *Web Spinner* application and go directly to the desktop to enter this information. You pull down the *Analyze* Menu and select *Basic Data* (or press CTRL + D) to activate the *Basic Data* panel. You then select 64,000 bits per second as the uniform link speed. You also enter 128 bits as the network packet size as shown in Figure C.2.

The next step in designing your network with *Web Spinner* is to create the ten network nodes on the desktop. You generate ten nodes with the *Node Pad* and position them on the screen so that they roughly approximate the relative geographic positions of the actual nodes. *Web Spinner* initially generates links on the screen that approximate the best geographic tree as you place new nodes on the desktop. This would be your "ideal" network if you were concerned only with the distances between the proposed nodes. Your initial network is shown in Figure C.3.

54
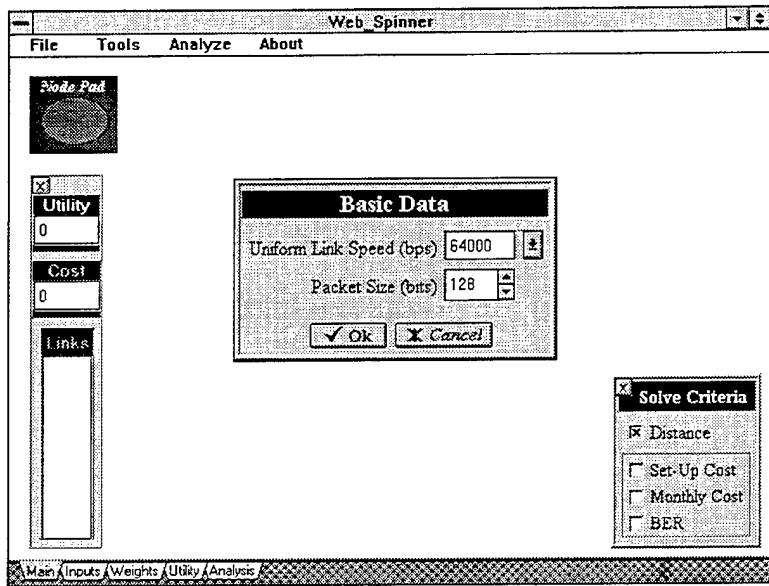
Figure C.2 Defining Initial Network Data.
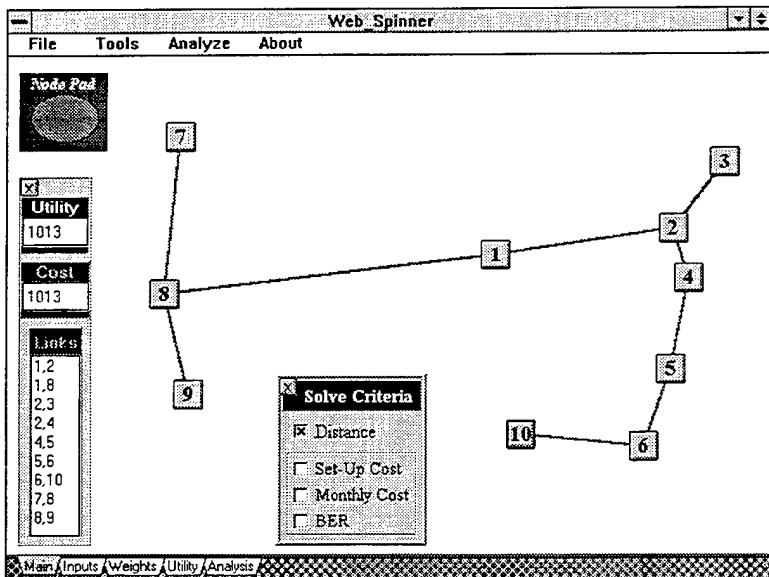
Figure C.3 Defining Network Nodes.

C. Link Data

Fortunately, the system administrator has already determined the data transmission requirements in packets per second for the proposed network. You select the *Inputs* tab along the bottom of the screen to go to the *Link Characteristics* screen. Select the first speed button

on the left side of the screen to activate the *Traffic Requirements* matrix. You enter the data as shown in Figure C.4.
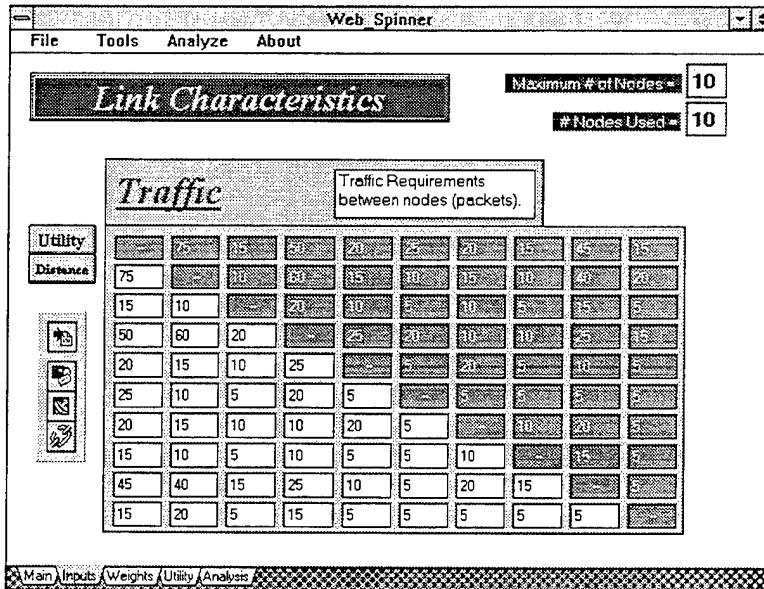
Figure C.4. Entering Traffic Requirement Data.

You then collect commercial tariff rates for leasing 64 KBPS links between every two node pairs in the proposed network. After some initial analysis, you select the best link for each node pair and enter the respective data into the *Set-Up Cost, Monthly* Cost, and *BER* matrices as shown in Figures C.5, C.6, and C.7.
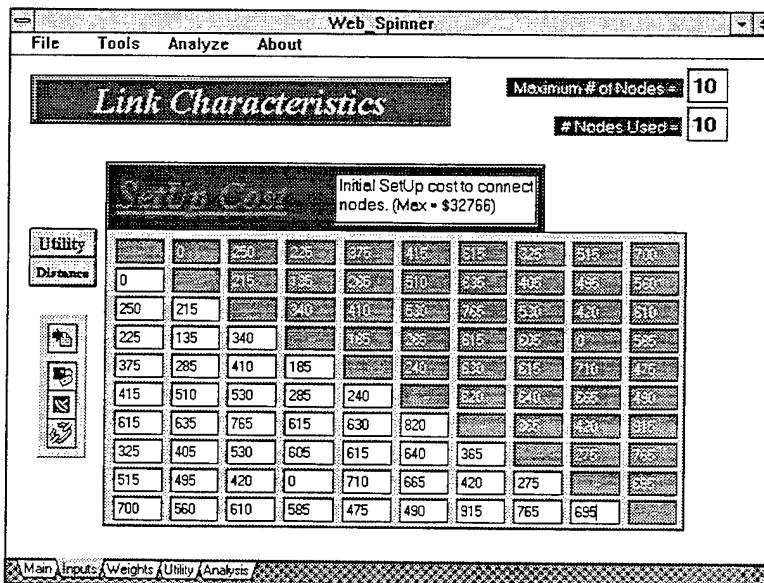
Figure C.5 Set-Up Cost Data.

Web_Spinner

File   Tools   Analyze   About

Link Characteristics

Maximum # of Nodes = 10
Nodes Used = 10

Monthly Cost   Monthly Costs to maintain each link. (Max = $32766)

Utility | Distance

100		90							
170	90								
120	75	185							
190	115	140	95						
210	135	190	200	140					
290	285	285	325	310	320				
220	270	290	365	280	230	140			
240	300	310	280	305	340	165	115		
265	215	245	170	190	185	230	210	255	

Main | Inputs | Weights | Utility | Analysis

Figure C.6 Monthly Cost Data.

Web_Spinner

File   Tools   Analyze   About

Link Characteristics

Maximum # of Nodes = 10
Nodes Used = 10

Error Rate   Error Rate of each Link. (10E-#)(negative values)

Utility | Distance

-10									
-8	-10								
-10	-10	-10							
-8	-9	-10	-10						
-8	-8	-9	-10	-9					
-9	-9	-9	-8	-8	-8				
-10	-10	-9	-10	-8	-9	-9			
-9	-8	-8	-9	-8	-8	-9	-10		
-7	-8	-8	-9	-8	-8	-7	-8	-8	

Main | Inputs | Weights | Utility | Analysis
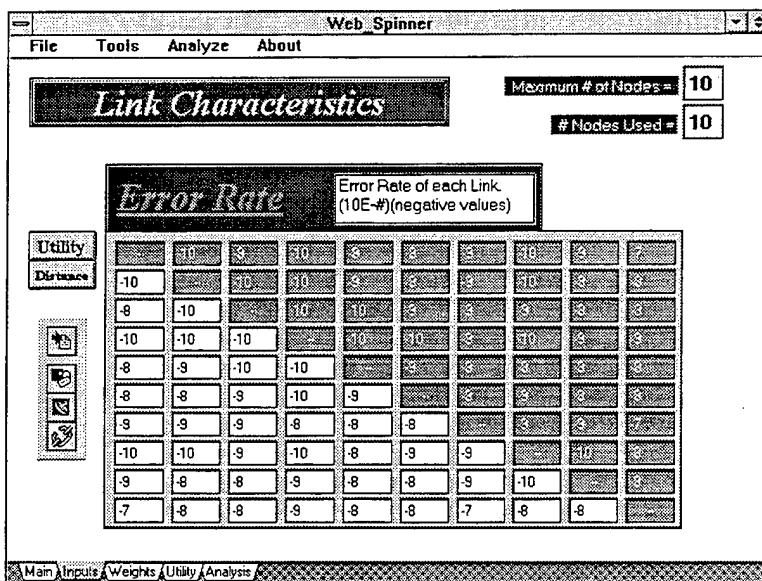
Figure C.7 Bit Error Rate Data.

D. Defining Criteria Weights

You then select the *Weights* tab to go to the *Weights* screen. Since minimal money is available to set-up the network, you assign this criteria a weight of 50%. In your initial tasking, you were instructed that the network should have a low error rate as it would be serving the fleet in

time of crises. Therefore, you assign the *BER* criteria a weight of 30%. You then finish the weight assignments by entering 20% in the *Monthly Cost* input box as shown in Figure C.8.
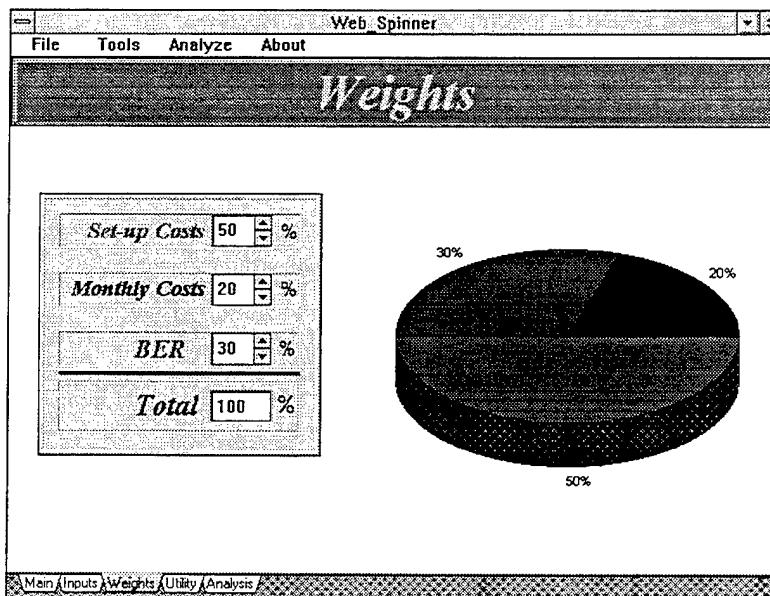
Figure C.8 Assignment of Criteria Weights.

E. Defining Criteria Utility Curves

Finally, you must help *Web Spinner* determine how to make value trade-offs between different levels of "goodness" for the given criteria. This is done on the *Web Spinner Utility* page which can be selected from the appropriate tab. You are presented with all three criteria initially graphed in a "risk neutral," linear curve. However, you decide that this is not representative of the true value trade-offs of your decision world -- especially for the *Set-Up Cost* and *BER* curves. You modify these curves until you are satisfied that they approximate how important each criteria and its associated cost are to you. You decide that you are not really concerned with *the Monthly Cost* criteria and leave it in its initial, linear form. (see Figure C.9)

F. Learning About Your Proposed Network

It is now time to begin learning how each of these criteria affect the network design. Return to the *Web Spinner* desktop. Select the *Solve Criteria* panel from the *Analysis* menu if it is not already displayed on the desktop. Check the *Set-Up Cost* box and *Web Spinner* redraws the network using *Set-Up Cost* as the sole selection criteria. Figure C.10 depicts this possible network.
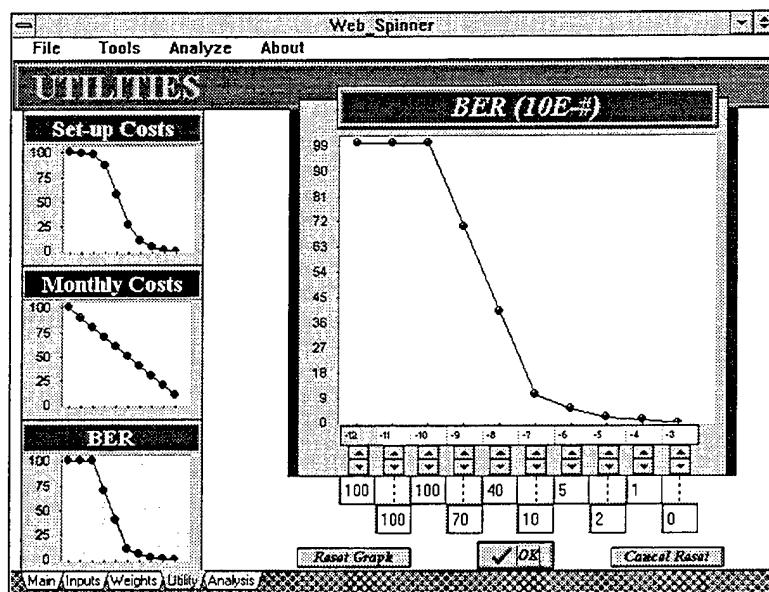
Figure C.9 Defining Utility Curves.
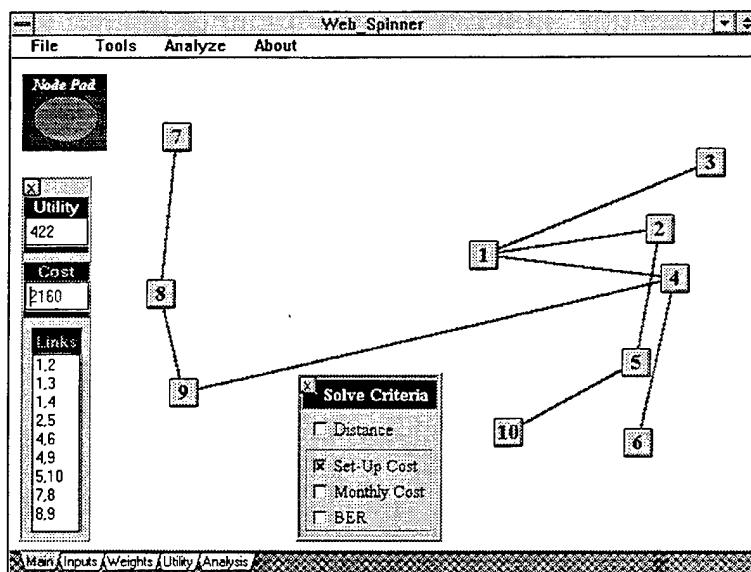
Figure C.10 Network Using Set-Up Costs Only.

You then look at the individual network designs using *Monthly Cost* and *BER* as the sole decision criteria by selecting the appropriate checkboxes. Since you believe that *Set-Up Costs* and *BER* are the most important criteria for deciding on the network design, you select both of these criteria and *Web Spinner* designs the network as depicted in Figure C.11. You then observe how each two criteria combination affects the final network design. Finally, you decide that all three criteria really should be used and check all three boxes. (see Figure C.12)

59
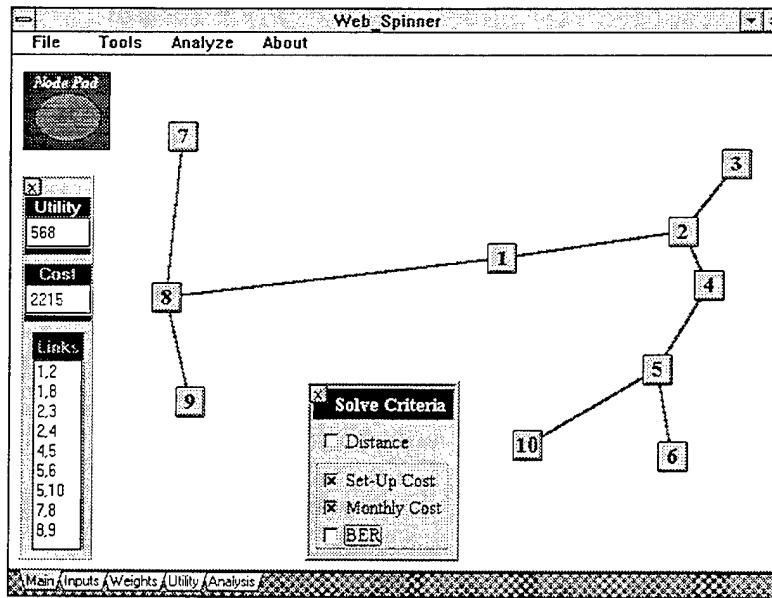
Figure C.11 Network Design Using Set-Up Costs and BER.
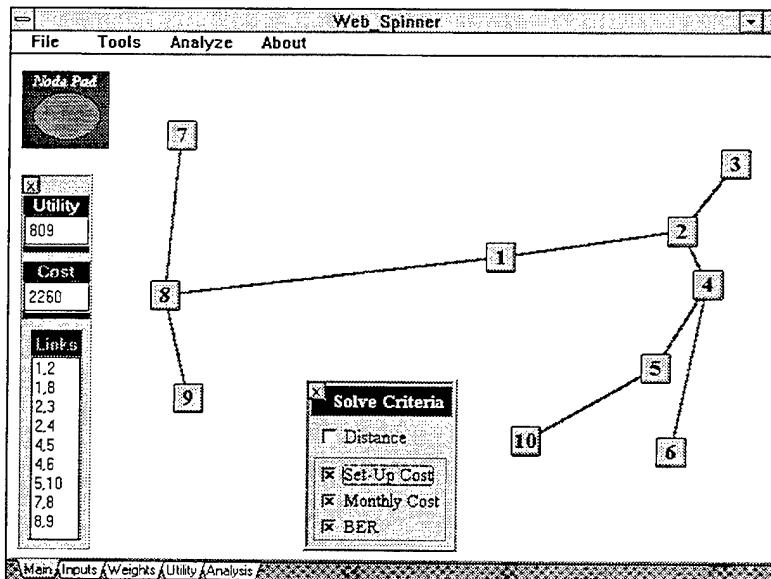
Figure C.12 Network Design Using All Three Criteria.

You then remember that you have already decided to use the previously established links between USTRANSCOM-Washington D.C. and Norfolk-San Diego. Since the current network design already incorporates the USTRANSCOM-Washington D.C. link, you simply click on that link to lock it into place. You then use the *Link Creator* tool under the *Tools* menu to create and lock a link between nodes four (Norfolk) and nine (San Diego) as shown in Figure C.13. *Web Spinner* immediately redraws the network to incorporate this link.
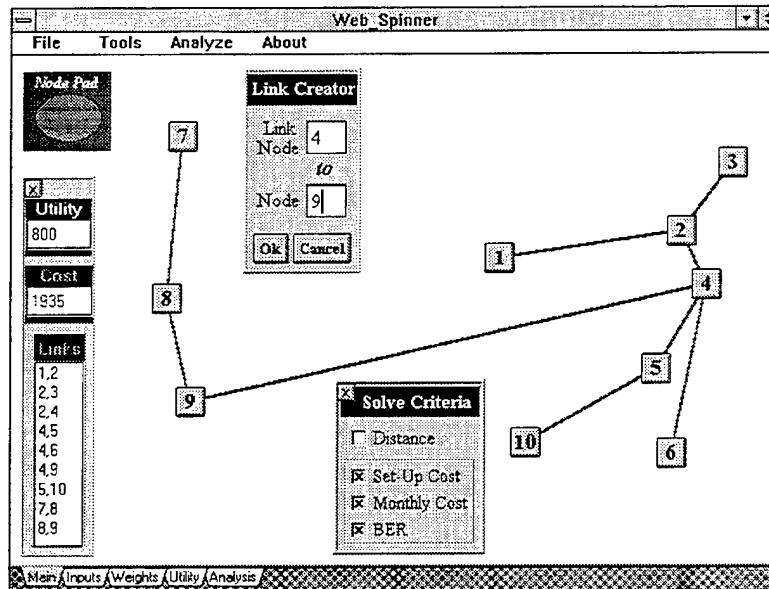
Figure C.13 Incorporating Already Established Links.

G. Incorporating New Data

The Budget Officer calls to inform you that Congress has appropriated more money to the USTRANSCOM O & M budget to cover unanticipated transportation costs incurred early in the fiscal year. This means that more money is now available to establish the network. However, he warns that this does not equate to a blank check as other departments also have unfunded requirements and will be competing for these funds. In addition, he advises that he no longer anticipates any additional funding in next years budget for new information system operations.

Armed with this new information, you decide to see how it affects the design of your network. You go to the *Web Spinner Weights* page and make some modifications. Since more money is now available to establish the network, you reduce the *Set-Up Costs* criteria to 15%. Concerned about minimizing future network costs, you increase the *Monthly Cost* criteria to 60%. Finally, you reduce the *BER* criteria weight to 25% to ensure the combined weights total 100% as seen in Figure C.14.

61
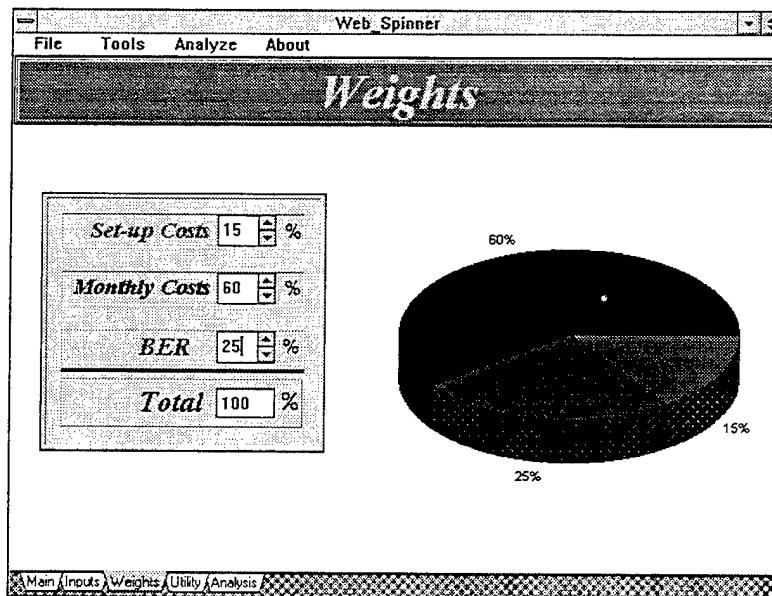
Figure C.14 Changing Weights to Reflect New Information.

You return to the *Web Spinner* desktop to see that the network has been redrawn to incorporate this new information. (see Figure C.15) The change in criteria weighting caused several links to change in the proposed network design.
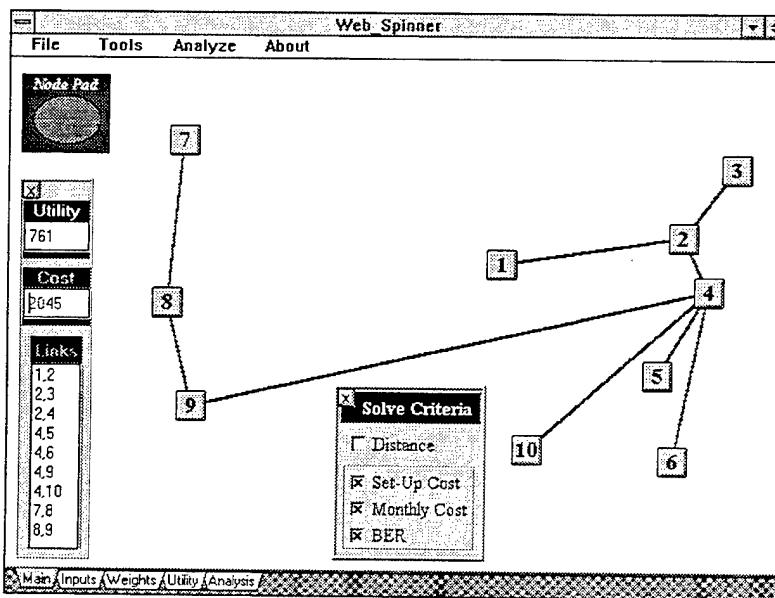
Figure C.15 Network Design Reflecting New Criteria Weighting.

62

Knowing that future year budgets are always subject to change, you decide to disregard the *Monthly Cost* criteria to see how this affects the network design. You deselect *the Monthly Cost* checkbox and two links change in the network as shown in Figure C.16.
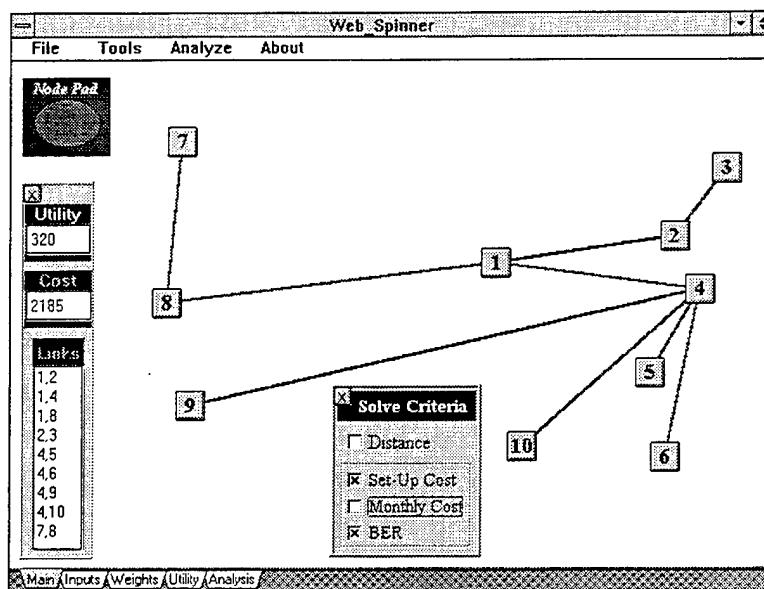
Figure C.16 New Network Disregarding Monthly Costs.

You also notice that the cost of establishing this network increases by more than seven percent when monthly costs are disregarded ($2185 as opposed to $2045). Therefore, you decide to re-include the *Monthly Cost* criteria. You decide that this is your "optimum" network based on all defined criteria. Figure C.17 represents your final network design.
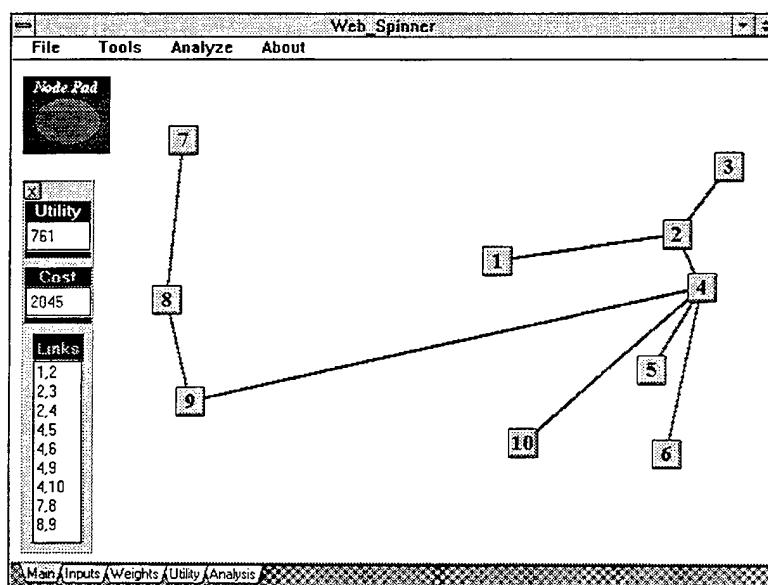
Figure C.17 Final Network Design.

H. Network Performance Analysis

You click on the *Analysis* tab to see the performance characteristics of your network design. Figure C.18 depicts this information.
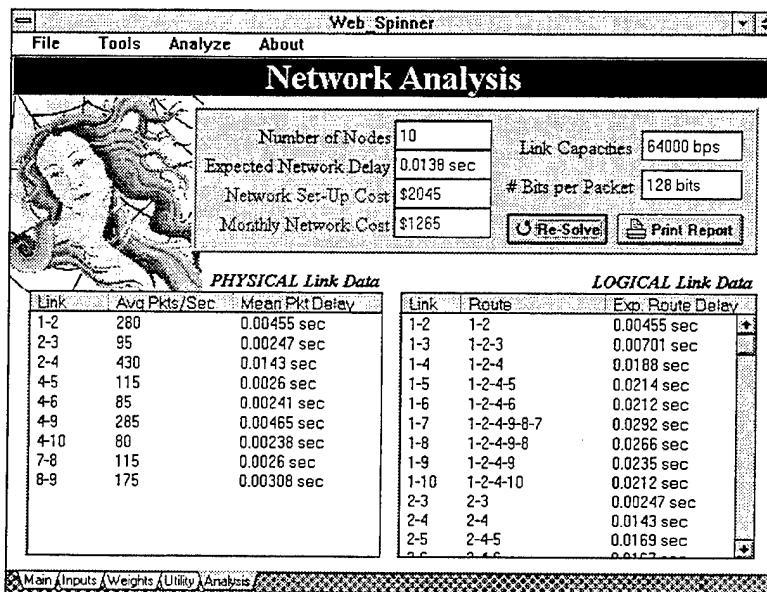
Figure C.18 Network Performance Characteristics.

I. Final Report

Pleased with your work, you click on the *Print Report* button to send a copy of your network design, performance analysis, and criteria information to the printer. Figures C.19 and C.20 show a copy of this report. (Note: Some printer drivers, particularly those associated with the Windows-95 operating system, are unable to print the two graphics at the top of the report and may leave those areas blank.) You make several copies of this report to present to the system administrator and your boss. You also archive a copy in the new systems file so that it can be referred to later to recreate and justify your motivations for selecting this particular network design.

Network Analysis

Number of Nodes: 10
Net Set-Up Cost: $2000
Monthly Net Cost: $1205

Expected Net Delay: 0.0142 sec
Link Capacities: 64,000 bps
Bits / Packet: 128 bits

Physical Link Data

Link	Avg Pkts/Sec	Mean Pkt Delay
1-2	280	0.00455 sec
2-3	95	0.00247 sec
2-4	430	0.0143 sec
4-5	190	0.00323 sec
4-9	285	0.00465 sec
4-10	80	0.00238 sec
5-6	85	0.00241 sec
7-8	115	0.0026 sec
8-9	175	0.00308 sec

Logical Link Data

Link	Route	Exp Route Delay
1-2	1-2	0.00455 sec
1-3	1-2-3	0.00701 sec
1-4	1-2-4	0.0188 sec
1-5	1-2-4-5	0.0221 sec
1-6	1-2-4-5-6	0.0245 sec
1-7	1-2-4-9-8-7	0.0292 sec
1-8	1-2-4-9-8	0.0266 sec
1-9	1-2-4-9	0.0235 sec
1-10	1-2-4-10	0.0212 sec
2-3	2-3	0.00247 sec
2-4	2-4	0.0143 sec
2-5	2-4-5	0.0175 sec
2-6	2-4-5-6	0.0199 sec
2-7	2-4-9-8-7	0.0246 sec
2-8	2-4-9-8	0.022 sec
2-9	2-4-9	0.0189 sec
2-10	2-4-10	0.0167 sec
3-4	3-2-4	0.0168 sec
3-5	3-2-4-5	0.02 sec
3-6	3-2-4-5-6	0.0224 sec
3-7	3-2-4-9-8-7	0.0271 sec
3-8	3-2-4-9-8	0.0245 sec
3-9	3-2-4-9	0.0214 sec
3-10	3-2-4-10	0.0191 sec
4-5	4-5	0.00323 sec
4-6	4-5-6	0.00564 sec
4-7	4-9-8-7	0.0103 sec
4-8	4-9-8	0.00773 sec
4-9	4-9	0.00465 sec
4-10	4-10	0.00238 sec
5-6	5-6	0.00241 sec
5-7	5-4-9-8-7	0.0136 sec
5-8	5-4-9-8	0.011 sec
5-9	5-4-9	0.00788 sec
5-10	5-4-10	0.00561 sec
6-7	6-5-4-9-8-7	0.016 sec
6-8	6-5-4-9-8	0.0134 sec
6-9	6-5-4-9	0.0103 sec
6-10	6-5-4-10	0.00802 sec
7-8	7-8	0.0026 sec
7-9	7-8-9	0.00567 sec
7-10	7-8-9-4-10	0.0127 sec
8-9	8-9	0.00308 sec
8-10	8-9-4-10	0.0101 sec
9-10	9-4-10	0.00703 sec

Figure C.19 Sample Report (page 1).

Traffic Data

1	2	3	4	5	6	7	8	9	10
–	75	15	50	20	25	20	15	45	15
75	–	10	60	15	10	15	10	40	20
15	10	–	20	10	5	10	5	15	5
50	60	20	–	25	20	10	10	25	15
20	15	10	25	–	5	20	5	10	5
25	10	5	20	5	–	5	5	5	5
20	15	10	10	20	5	–	10	20	5
15	10	5	10	5	5	10	–	15	5
45	40	15	25	10	5	20	15	–	5
15	20	5	15	5	5	5	5	5	–

Bit Error Rates

1	2	3	4	5	6	7	8	9	10
–	–10	–8	–10	–8	–8	–9	–10	–9	–7
–10	–	–10	–10	–9	–8	–9	–10	–8	–8
–8	–10	–	–10	–10	–9	–9	–9	–8	–8
–10	–10	–10	–	–10	–10	–8	–10	–9	–9
–8	–9	–10	–10	–	–9	–8	–8	–8	–8
–8	–8	–9	–10	–9	–	–8	–9	–8	–8
–9	–9	–9	–8	–8	–8	–	–9	–9	–7
–10	–10	–9	–10	–8	–9	–9	–	–10	–8
–9	–8	–8	–9	–8	–8	–9	–10	–	–8
–7	–8	–8	–9	–8	–8	–7	–8	–8	–

Link Set-Up Costs

1	2	3	4	5	6	7	8	9	10
–	0	250	225	375	415	615	325	515	700
0	–	215	135	285	510	635	405	495	560
250	215	–	340	410	530	765	530	420	610
225	135	340	–	185	285	615	605	0	585
375	285	410	185	–	240	630	615	710	475
415	510	530	285	240	–	820	640	665	490
615	635	765	615	630	820	–	365	420	915
325	405	530	605	615	640	365	–	275	756
515	495	420	0	710	665	420	275	–	695
700	560	610	585	475	490	915	765	695	–

Monthly Link Costs

1	2	3	4	5	6	7	8	9	10
–	100	170	120	190	210	290	220	240	265
100	–	90	75	115	135	285	270	300	215
170	90	–	185	140	190	285	290	310	245
120	75	185	–	95	200	325	265	280	170
190	115	140	95	–	140	310	280	305	190
210	135	190	200	140	–	320	230	340	185
290	285	285	325	310	320	–	140	165	230
220	270	290	265	280	230	140	–	115	210
240	300	310	280	305	340	165	115	–	255
265	215	245	170	190	185	230	210	255	–

Criteria Used for Network Generation

Set-Up Cost   Monthly Cost   BER

Set-Up Cost Weight =   15 %
Monthly Cost Weight =   60 %
BER Cost Weight   =   25 %

Figure C.20 Sample Report (page 2).

APPENDIX D. DELPHI® AND PROGRAM CODE

A description of the Borland Delphi 1.0 programming environment is provided along with a complete *Web Spinner* source code listing. These files are included on the *Web Spinner* Source Disk along with a copy of the Delphi TLineDraw component designed by Blain R. Southam of Brigham Young University. (The TLineDraw component is considered freeware and may be freely distributed provided due credit is given to the author.)

A. DELPHI®

Web Spinner was developed in the Borland Delphi 1.0 programming environment. It uses an object-oriented version of the Pascal programming language called Object Pascal. Delphi is a complex programming environment that incorporates standard Microsoft Windows component libraries, Borland Paradox database objects, enhanced graphic tool components, and a fast compiler into a fourth generation programming package. The compiler generates a stand-alone executable file (*.exe) which can be executed on any Microsoft Windows-supported system.

Delphi was selected to implement the *Web Spinner* DSS on account of it object-oriented approach, incorporation of graphical components, and the author's familiarity with the basic Pascal language.

B. *Web Spinner* Code Listing

Web Spinner was implemented using three Object-Pascal code units: Web_Spnr.dpr; WSAbout.pas and Web1.pas. Web_Spnr.dpr is the main program module which uses the remaining two units during the execution of the program. A complete listing of the three modules is provided.

1. Web_Spnr.dpr

```
{*********************************************************************
**    Program:   Webspnr.dpr                                       **
**    Author:    Jeffrey A. Margraf                                **
**    Updated:   02 August 1996                                    **
**    Uses:      WSAbout.pas                                       **
**               Web1.pas                                          **
**                                                                 **
**    Description: This is the main program which launches the     **
**                 Web Spinner decision support system.            **
**                                                                 **
*********************************************************************}

program Webspnr;

uses
  Forms,
  Wsabout in 'WSABOUT.PAS',    {About Information}
  Web1 in 'WEB1.PAS';          {Web_Spinner DSS}

{$R *.RES}

begin
  Application.CreateForm(TAbout, About);
  Application.CreateForm(TWeb_Spinner, Web_Spinner);
  Application.Run
end.
```

2. WSAbout.pas

```
{*********************************************************************
**                                                                 **
**    Module:    WSAbout.pas                                       **
**    Author:    Jeffrey A. Margraf                                **
**    Updated:   02 August 1996                                    **
**    Used by:   Webspnr.dpr                                       **
**    Uses:      Web1.pas                                          **
**                                                                 **
**    Description: This module contains all the Web Spinner basic  **
**                 program information found on the "About" page.  **
**                                                                 **
*********************************************************************}

unit WSAbout;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls, Buttons;

type
  TAbout = class(TForm)
```

```
      BitBtn1: TBitBtn;
      BitBtn2: TBitBtn;
      BitBtn3: TBitBtn;
      Button1: TButton;
      Image1: TImage;
      Image2: TImage;
      Image3: TImage;
      Label1: TLabel;
      Label2: TLabel;
      Label3: TLabel;
      Label4: TLabel;
      Memo1: TMemo;
      Memo2: TMemo;
      Memo3: TMemo;
      Memo4: TMemo;
      Panel1: TPanel;
      Shape1: TShape;
      Shape2: TShape;
      Shape3: TShape;
      Shape4: TShape;
      Shape5: TShape;

      procedure Button1Click(Sender: TObject);
      procedure Image2Click(Sender: TObject);
      procedure BitBtn1Click(Sender: TObject);
      procedure BitBtn2Click(Sender: TObject);
      procedure BitBtn3Click(Sender: TObject);

   private
      { Private declarations }

   public
      { Public declarations }

   end;

var
   About: TAbout;
   Start: Boolean;

implementation
{$R *.DFM}

uses Web1;

procedure TAbout.Button1Click(Sender: TObject);
begin
   Web_Spinner.Show
end;

procedure TAbout.Image2Click(Sender: TObject);
begin
   BitBtn3.Visible := false;
   Panel1.Visible := true
end;

procedure TAbout.BitBtn1Click(Sender: TObject);
```

```
begin
  Panel1.Visible := false;
  BitBtn3.Visible := true
end;

procedure TAbout.BitBtn2Click(Sender: TObject);
begin
  BitBtn1.Visible := false;
  BitBtn2.Visible := false;
  Print;
  BitBtn1.Visible := true;
  BitBtn2.Visible := true
 end;

procedure TAbout.BitBtn3Click(Sender: TObject);
begin
  BitBtn3.Visible := false;
  Print;
  BitBtn3.Visible := true
end;

end.
```

3. Web1.pas

```
{*********************************************************************
**                                                                 **
**   Module:    Web1.pas                                           **
**   Author:    Jeffrey A. Margraf                                 **
**   Updated:   02 August 1996                                     **
**   Used by:   Web_Spnr.dpr                                       **
**   Uses:      WSAbout.pas                                        **
**                                                                 **
**   Description: This module contains all the technical           **
**                functionality associated with the Web Spinner DSS. **
**                                                                 **
*********************************************************************}

unit Web1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls, Tabs, StdCtrls, Buttons, VBXCtrl, Line,
  Menus, Bigauge, Grids, Chart2fx, ChartFX, Spin;

const
  Infinity = 32767;        {Artificial Infinity Value = Max Integer}
  NumNodes = 10;           {Maximum Number of Nodes}
  Spacer   = #1;
  Spacer2  = #2;
```

```
type
   LinkType = (NoLink, Linked, LockLinked, Redundant, Testing, Checking,
               Used, LUsed);
   InputPanelType = (SetUpCost, MonthlyCost, BER, Traffic, Distance,
                     Utility);

   LinkMatrixType      = Array[1..NumNodes, 1..NumNodes] of LinkType;
   InputMatrixType     = Array[1..NumNodes, 1..NumNodes] of TEdit;
   MasterMatrixType    = Array[1..NumNodes, 1..NumNodes] of Integer;
   RouteMatrixType     = Array[1..NumNodes, 1..NumNodes] of String[3 *
                         NumNodes];
   ExpDelayMatrixType = Array[1..NumNodes, 1..NumNodes] of Real;

   NodeArrayType       = Array[1..NumNodes] of TBitBtn;
   LinkArrayType       = Array[1..NumNodes] of TLineDraw;
   UtilityGraphType    = Array[1..10] of Integer;


{*********************************************************************
**   These are the visual components (objects) available on the    **
**   Web Spinner screens.                                          **
*********************************************************************}

TWeb_Spinner = class(TForm)
     About1: TMenuItem;
     About2: TMenuItem;
     About3: TMenuItem;
     About5: TMenuItem;
     About6: TMenuItem;
     BasicData2: TMenuItem;
     Bevel1: TBevel;
     Bevel2: TBevel;
     Bevel3: TBevel;
     Bevel4: TBevel;
     BitBtn2: TBitBtn;
     BitBtn3: TBitBtn;
     BitBtn4: TBitBtn;
     BitBtn5: TBitBtn;
     BitBtn6: TBitBtn;
     BitBtn7: TBitBtn;
     BitBtn8: TBitBtn;
     BitBtn9: TBitBtn;
     BitBtn10: TBitBtn;
     BitBtn12: TBitBtn;
     BitBtn13: TBitBtn;
     BitBtn14: TBitBtn;
     BitBtn15: TBitBtn;
     Button1: TButton;
     ChartFX1: TChartFX;
     ChartFX2: TChartFX;
     ChartFX3: TChartFX;
     ChartFX4: TChartFX;
     ChartFX6: TChartFX;
     CheckBox1: TCheckBox;
     CheckBox2: TCheckBox;
     CheckBox3: TCheckBox;
     CheckBox4: TCheckBox;
```

```
Close1: TMenuItem;
ComboBox1: TComboBox;
Edit1: TEdit;
Edit2: TEdit;
Edit3: TEdit;
Edit4: TEdit;
Edit5: TEdit;
Edit6: TEdit;
Edit7: TEdit;
Edit8: TEdit;
Edit9: TEdit;
Edit10: TEdit;
Edit11: TEdit;
Edit12: TEdit;
Edit13: TEdit;
Edit14: TEdit;
Edit15: TEdit;
Edit16: TEdit;
Edit17: TEdit;
Edit18: TEdit;
Edit19: TEdit;
Edit20: TEdit;
Edit21: TEdit;
Edit22: TEdit;
Edit23: TEdit;
File1: TMenuItem;
GroupBox1: TGroupBox;
Header1: THeader;
Header2: THeader;
Image1: TImage;
Image2: TImage;
Image3: TImage;
Image4: TImage;
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Label7: TLabel;
Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
Label13: TLabel;
Label14: TLabel;
Label15: TLabel;
Label16: TLabel;
Label17: TLabel;
Label18: TLabel;
Label19: TLabel;
Label20: TLabel;
Label21: TLabel;
Label22: TLabel;
Label23: TLabel;
Label27: TLabel;
```

```
Label29: TLabel;
Label30: TLabel;
Label31: TLabel;
Label32: TLabel;
Label33: TLabel;
Label34: TLabel;
Label35: TLabel;
Label36: TLabel;
Label37: TLabel;
LineDraw1: TLineDraw;
LineDraw3: TLineDraw;
LineDraw4: TLineDraw;
LineDraw5: TLineDraw;
LineDraw6: TLineDraw;
LineDraw7: TLineDraw;
LineDraw8: TLineDraw;
LineDraw9: TLineDraw;
LinkInfoBox1: TMenuItem;
LinkLegend1: TMenuItem;
ListBox1: TListBox;
ListBox2: TListBox;
ListBox3: TListBox;
ListBox4: TListBox;
MainMenu1: TMainMenu;
N2: TMenuItem;
N3: TMenuItem;
Notebook1: TNotebook;
Panel1: TPanel;
Panel2: TPanel;
Panel3: TPanel;
Panel4: TPanel;
Panel5: TPanel;
Panel6: TPanel;
Panel7: TPanel;
Panel8: TPanel;
Panel9: TPanel;
Panel10: TPanel;
Panel11: TPanel;
Panel12: TPanel;
Panel13: TPanel;
Panel14: TPanel;
Panel15: TPanel;
Panel16: TPanel;
Panel17: TPanel;
Panel18: TPanel;
Panel19: TPanel;
Panel20: TPanel;
Panel21: TPanel;
Panel22: TPanel;
Panel23: TPanel;
Panel24: TPanel;
Panel25: TPanel;
Panel28: TPanel;
Panel29: TPanel;
Panel30: TPanel;
Panel31: TPanel;
Print1: TMenuItem;
```

```
    Shape1: TShape;
    Shape2: TShape;
    Shape3: TShape;
    Solve1: TMenuItem;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    SpinButton1: TSpinButton;
    SpinButton2: TSpinButton;
    SpinButton3: TSpinButton;
    SpinButton4: TSpinButton;
    SpinButton5: TSpinButton;
    SpinButton6: TSpinButton;
    SpinButton7: TSpinButton;
    SpinButton8: TSpinButton;
    SpinButton9: TSpinButton;
    SpinButton10: TSpinButton;
    SpinEdit1: TSpinEdit;
    SpinEdit2: TSpinEdit;
    SpinEdit3: TSpinEdit;
    SpinEdit4: TSpinEdit;
    StringGrid1: TStringGrid;
    TabSet1: TTabSet;


{**********************************************************************
**   These are the procedures (methods) that work with the Web      **
**   Spinner visual components.                                     **
**********************************************************************}

    procedure TabSet1Change(Sender: TObject; NewTab: Integer;
                        var AllowChange: Boolean);
    procedure FormActivate(Sender: TObject);
    procedure GenericMouseDown(Sender: TObject;
                            Button: TMouseButton; Shift: TShiftState;
                            X, Y: Integer);
    procedure MovePanel(var Box: TPanel; X, Y: Integer);
    procedure MoveBitBtn(var Node: TBitBtn; X, Y: Integer);
    procedure GenericMouseMove(Sender: TObject; Shift: TShiftState;
                            X, Y: Integer);
    procedure GenericMouseUp(Sender: TObject; Button: TMouseButton;
                            Shift: TShiftState; X, Y: Integer);
    procedure InitializeLinks(var Connections: LinkMatrixType);
    procedure LockTheLink(Sender: TObject; Button: TMouseButton;
                        Shift: TShiftState; X, Y: Integer);
    procedure InitializeMatrix(var DataMatrix: MasterMatrixType;
                                Data : Integer; Start: Integer);
    procedure FillInputMatrix(DataMatrix: MasterMatrixType;
                        var InputMatrix: InputMatrixType);
    function DistanceBetweenPoints(Point1, Point2: TPoint): Longint;
    procedure DistanceIsCost(var DistanceMatrix: MasterMatrixType;
                                Nodes: NodeArrayType);
    procedure ExtractInputs(var DataMatrix: MasterMatrixType;
                            InputMatrix: InputMatrixType);
    function Interpolate(Input, XAxisLeft, XAxisRight, ValueLeft,
                        ValueRight: Integer): Real;
```

```
function GetUtility(Input: Integer; InputType: InputPanelType):
                       Integer;
procedure FillUtilityMatrix(var UtilityMatrix: MasterMatrixType;
                                SetUpCostMatrix, MonthlyCostMatrix,
                                BERMatrix: MasterMatrixType;
                                SetUpCostWt, MonthlyCostWt,
                                BERWt: Integer);
procedure SolveMinTree(var DataMatrix: MasterMatrixType;
                           ExpenseMatrix: MasterMatrixType;
                       var Connections: LinkMatrixType);
procedure SolveMaxTree(var DataMatrix: MasterMatrixType;
                           ExpenseMatrix: MasterMatrixType;
                       var Connections: LinkMatrixType);
procedure DrawLinks(Connections: LinkMatrixType;
                    Nodes: NodeArrayType;
                var Links: LinkArrayType);
function ComputeTotal(DataMatrix: MasterMatrixType;
                      Connections: LinkMatrixType): Integer;
procedure Main(Sender: TObject);
procedure Shape1MouseDown(Sender: TObject; Button: TMouseButton;
                              Shift: TShiftState; X, Y: Integer);
procedure BitBtn2Click(Sender: TObject);
procedure Print1Click(Sender: TObject);
procedure About1Click(Sender: TObject);
procedure SpeedButton1Click(Sender: TObject);
procedure SpeedButton4Click(Sender: TObject);
procedure SpeedButton2Click(Sender: TObject);
procedure Notebook1PageChanged(Sender: TObject);
procedure OnExitInputBox(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure TotalWeights(Sender: TObject);
procedure Tools1Click(Sender: TObject);
procedure BitBtn5Click(Sender: TObject);
procedure BitBtn4Click(Sender: TObject);
procedure SwitchGraphValues(var UtilArray: UtilityGraphType;
                                var SmallChart: TChartFX;
                                MainChart: TChartFX);
procedure ChartFX2LButtonDblClk(Sender: TObject; var X, Y, nSerie,
                                nPoint, nRes: Integer);
procedure ChartFX3LButtonDblClk(Sender: TObject; var X, Y, nSerie,
                                nPoint, nRes: Integer);
procedure ChartFX4LButtonDblClk(Sender: TObject; var X, Y, nSerie,
                                nPoint, nRes: Integer);
procedure GenericSBDownClick(var EditBox: TEdit);
procedure GenericSBUpClick(var EditBox: TEdit);
procedure SpinButton1DownClick(Sender: TObject);
procedure SpinButton1UpClick(Sender: TObject);
procedure ResetGraph(var Chart: TChartFX);
procedure BitBtn6Click(Sender: TObject);
procedure BitBtn7Click(Sender: TObject);
procedure LinkInfoBox1Click(Sender: TObject);
procedure BitBtn3Click(Sender: TObject);
procedure BitBtn8Click(Sender: TObject);
procedure SpeedButton3Click(Sender: TObject);
procedure BitBtn9Click(Sender: TObject);
procedure CheckBox4Click(Sender: TObject);
```

```pascal
    procedure CheckBox1Click(Sender: TObject);
    procedure BitBtn10Click(Sender: TObject);
    procedure Solve1Click(Sender: TObject);
    procedure BasicData2Click(Sender: TObject);
    procedure BitBtn12Click(Sender: TObject);
    procedure LinkLegend1Click(Sender: TObject);
    procedure FindLoops(Node1, Node2, Parent, SuperParent: Integer;
                   var Route: String);
    procedure FillRouteMatrix;
    procedure ComputeAvgLinkActy;
    procedure FillExpectedDelayMatrix;
    procedure Close1Click(Sender: TObject);
    procedure BitBtn13Click(Sender: TObject);
    procedure ListBox3DrawItem(Control: TWinControl; Index: Integer;
                              Rect: TRect; State: TOwnerDrawState);
    procedure Header1Sizing(Sender: TObject; ASection, AWidth: Integer);
    procedure ListBox4DrawItem(Control: TWinControl; Index: Integer;
                              Rect: TRect; State: TOwnerDrawState);
    procedure Header2Sizing(Sender: TObject; ASection, AWidth: Integer);
    procedure BitBtn14Click(Sender: TObject);
    procedure BitBtn15Click(Sender: TObject);
    procedure PrintLine(Items: TStringList);
    procedure PrintHeader;
    procedure PrintNetworkInfo;
    procedure PrintColNames;
    procedure PrintData;
    procedure PrintCriteria;

  private
    { Private declarations }
  public
    { Public declarations }
  end;


{*******************************************************************
 **   These are the Web Spinner global variables.              **
 *******************************************************************}

var
  Web_Spinner: TWeb_Spinner;
  Connections: LinkMatrixType;
  InputMatrix: InputMatrixType;
  UtilityMatrix: MasterMatrixType;
  DistanceMatrix: MasterMatrixType;
  SetUpCostMatrix, MonthlyCostMatrix, BERMatrix: MasterMatrixType;
  TrafficMatrix: MasterMatrixType;
  Nodes: NodeArrayType;
  Links: LinkArrayType;
  UtilSetUpCostArray, UtilMonthlyCostArray, UtilBERArray:
      UtilityGraphType;
  UtilSetUpCostXAxis, UtilMonthlyCostXAxis, UtilBERXAxis:
      UtilityGraphType;
  InputPanel, UtilityGraph: InputPanelType;
  MouseOrigin: TPoint;
  Moving, DrawingLink, Found: Boolean;
  Path: String;
```

76

```
    RouteMatrix : RouteMatrixType;
    AvgLinkActyMatrix: MasterMatrixType;
    ExpDelayMatrix: ExpDelayMatrixType;
    LinkSpeed, PacketSize: LongInt;
    RedundantRoutes: Integer;
    NumUsedNodes, TotalUtil, TotalCost: Integer;
    SetUpCostUtil, MonthlyCostUtil, BERUtil: Integer;
    SetUpCostWt, MonthlyCostWt, BERWt: Integer;
    PixelsInInchX, TenthsOfInchPixelsY, LineHeight, AmountPrinted:
        Integer;

implementation
{$R *.DFM}

uses WSAbout, Printers;


{***************************************************************
 **   This procedure provides the "tabbed notebook" functionality   **
 **   of the program.                                               **
 ***************************************************************}

procedure TWeb_Spinner.TabSet1Change(Sender: TObject; NewTab: Integer;
                                     var AllowChange: Boolean);
begin
    Notebook1.PageIndex := NewTab;
    If Notebook1.PageIndex = 1 then BitBtn9Click(Sender);
    If Notebook1.PageIndex = 4 then BitBtn14Click(Sender)
end;


{***************************************************************
 **   This procedure is executed when the application is launched.   **
 **   It establishes the initial conditions of many global variables, **
 **   creates background network nodes and links, and initializes   **
 **   all of the Web Spinner data matrices and arrays.              **
 ***************************************************************}

procedure TWeb_Spinner.FormActivate(Sender: TObject);
var
  Loop, Count, Tab: Integer;
begin
  Panel18.Parent := Web_Spinner;
  Panel18.Visible := false;
  LinkSpeed := 9600;
  PacketSize := 128;

  NumUsedNodes := 0;
  Edit2.Text := IntToStr(NumNodes);
  Edit3.Text := IntToStr(NumUsedNodes);
  InputPanel := Distance;
  Panel6.Height := (NumNodes * 24) + 10;
  Panel6.Width := (NumNodes * 48) + 8;
  UtilityGraph := SetUpCost;

  InitializeLinks(Connections);
  InitializeMatrix(SetUpCostMatrix, 0, 1);
```

```
InitializeMatrix(MonthlyCostMatrix, 0, 1);
InitializeMatrix(BERMatrix, 0, 1);
InitializeMatrix(TrafficMatrix, 0, 1);
InitializeMatrix(UtilityMatrix, 0, 1);
TotalWeights(Sender);

for Loop := 1 to NumNodes do begin

  Nodes[Loop] := TBitBtn.Create(self);
  with Nodes[Loop] do begin
    Parent := Panel5;
    Show;
    Visible := false;
    Top := Panel1.Top + 24;
    Left := Panel1.Left + 24;
    Height := 25;
    Width := 25;
    Caption := IntToStr(Loop);
    Font.Color := clBlue;
    ShowHint := true;
    OnMouseDown := GenericMouseDown;
    OnMouseMove := GenericMouseMove;
    OnMouseUp := GenericMouseUp
  end;

  Links[Loop] := TLineDraw.Create(self);
  with Links[Loop] do begin
    Parent := Panel5;
    Show;
    Visible := false;
    Top:= Panel1.Top;
    Left:= Panel1.Left;
    Height := 25;
    Width := 25;
    Shape := stTopLine;
    Pen.Width := 2;
    Pen.Color := clFuchsia;
    OnMouseDown := LockTheLink;
    Hint := 'Link ' + IntToStr(Loop);
    ShowHint := false
  end;

  Tab := 0;
  for Count := 1 to NumNodes do begin
    InputMatrix[Loop, Count] := TEdit.Create(self);
    with InputMatrix[Loop, Count] do begin
      Parent := Panel6;
      Show;
      Visible := true;
      Top:= 8 + ((Loop - 1) * 24);
      Left:= 8 + ((Count - 1) * 48);
      Height := 20;
      Width := 41;
      Font.Size := 9;
      Text := '       --';
      OnExit := OnExitInputBox;
      Hint := 'Nodes ' + IntToStr(Loop) + ' & ' + IntToStr(Count);
```

```
        ShowHint := true;
        if Loop <= Count then begin
          Color := clBlack;
          Font.Color := clWhite;
          ReadOnly := true;
          TabOrder := Tab;
          TabStop := true;
          Tab := Tab + 1
        end
      end
    end
end;

for Loop := 1 to 10 do begin
  UtilSetUpCostXAxis[Loop] := Loop * 100;
  UtilSetUpCostArray[Loop] := (11 - Loop) * 10;
  UtilMonthlyCostXAxis[Loop] := Loop * 50;
  UtilMonthlyCostArray[Loop] :=(11 - Loop) * 10;
  UtilBERXAxis[Loop] := ((11 - Loop) + 2) * (-1);
  UtilBERArray[Loop] := (11 - Loop) * 10
end;

ResetGraph(ChartFX2);
ResetGraph(ChartFX3);
ResetGraph(ChartFX4);
ResetGraph(ChartFX6);

ChartFX2.OpenData[COD_VALUES] := MakeLong(1,10);
ChartFX6.OpenData[COD_VALUES] := MakeLong(1,10);
ChartFX2.ThisSerie := 0;
ChartFX6.ThisSerie := 0;
for Loop := 0 to 9 do begin
  ChartFX2.Value[Loop] := UtilSetUpCostArray[Loop + 1];
  ChartFX6.Value[Loop] := UtilSetUpCostArray[Loop + 1];
  StringGrid1.Cells[Loop, 0] := IntToStr(UtilSetUpCostXAxis[Loop + 1])
end;
ChartFX2.CloseData[COD_VALUES] := 0;
ChartFX6.CloseData[COD_VALUES] := 0;

ChartFX3.OpenData[COD_VALUES] := MakeLong(1,10);
ChartFX3.ThisSerie := 0;
for Loop := 0 to 9 do
  ChartFX3.Value[Loop] := UtilBERArray[Loop + 1];
ChartFX3.CloseData[COD_VALUES] := 0;

ChartFX4.OpenData[COD_VALUES] := MakeLong(1,10);
ChartFX4.ThisSerie := 0;
for Loop := 0 to 9 do
  ChartFX4.Value[Loop] := UtilMonthlyCostArray[Loop + 1];
ChartFX4.CloseData[COD_VALUES] := 0;

Edit4.Text := IntToStr(UtilSetUpCostArray[1]);
Edit5.Text := IntToStr(UtilSetUpCostArray[2]);
Edit6.Text := IntToStr(UtilSetUpCostArray[3]);
Edit7.Text := IntToStr(UtilSetUpCostArray[4]);
Edit12.Text := IntToStr(UtilSetUpCostArray[5]);
Edit13.Text := IntToStr(UtilSetUpCostArray[6]);
```

```
   Edit14.Text := IntToStr(UtilSetUpCostArray[7]);
   Edit15.Text := IntToStr(UtilSetUpCostArray[8]);
   Edit16.Text := IntToStr(UtilSetUpCostArray[9]);
   Edit17.Text := IntToStr(UtilSetUpCostArray[10]);

   ChartFX2.RGB2DBk := clYellow;
   Panel5.Visible := true;
   Panel6.Visible := true;
   Notebook1.PageIndex := 0
end;


{****************************************************************
 **   The next five procedures are custom-designed, "generic" routines **
 **   that allow the user to move (drag) visual components around with **
 **   the mouse.  The GenericMouseDown procedure also allows the user  **
 **   to delete nodes and then compacts data matrices.                 **
 ****************************************************************}

procedure TWeb_Spinner.GenericMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var
  Loop, ShiftLoop, Row, Column : Integer;
begin
   if Button = mbLeft then begin
     MouseOrigin := Point(X, Y);
     Moving := true
   end
   else if Button = mbRight then
     for Loop := 1 to NumUsedNodes do
       if Sender = Nodes[Loop] then begin
         if MessageDlg('Delete Node ' + Nodes[Loop].Caption + '?',
                       mtConfirmation, mbOkCancel, 0) = mrOk then
                       begin
                         if Loop < NumUsedNodes then begin
                         Nodes[Loop].Top := Nodes[NumUsedNodes].Top;
                         Nodes[Loop].Left := Nodes[NumUsedNodes].Left;

                         for ShiftLoop := 1 to NumUsedNodes do
                           if ShiftLoop <> Loop then begin
                             Connections[Loop, ShiftLoop] :=
                                 Connections[NumUsedNodes, ShiftLoop];
                             Connections[ShiftLoop, Loop] :=
                                 Connections[Loop, ShiftLoop];

                             Connections[NumUsedNodes, ShiftLoop] :=
                                 NoLink;
                             Connections[ShiftLoop, NumUsedNodes] :=
                                 NoLink;

                             SetUpCostMatrix[Loop, ShiftLoop] :=
                              SetUpCostMatrix[NumUsedNodes, ShiftLoop];
                             SetUpCostMatrix[ShiftLoop, Loop] :=
                              SetUpCostMatrix[Loop, ShiftLoop];

                             MonthlyCostMatrix[Loop, ShiftLoop] :=
                             MonthlyCostMatrix[NumUsedNodes, ShiftLoop];
```

80

```
                                    MonthlyCostMatrix[ShiftLoop, Loop] :=
                                    MonthlyCostMatrix[Loop, ShiftLoop];

                                    BERMatrix[Loop, ShiftLoop] :=
                                            BERMatrix[NumUsedNodes, ShiftLoop];
                                    BERMatrix[ShiftLoop, Loop] :=
                                            BERMatrix[Loop, ShiftLoop];

                                    TrafficMatrix[Loop, ShiftLoop] :=
                                        TrafficMatrix[NumUsedNodes, ShiftLoop];
                                    TrafficMatrix[ShiftLoop, Loop] :=
                                        TrafficMatrix[Loop, ShiftLoop]
                                end;
                            end;

                            Nodes[NumUsedNodes].Visible := false;
                            Nodes[NumUsedNodes].Top := Panel1.Top + 24;
                            Nodes[NumUsedNodes].Left := Panel1.Left + 24;
                            NumUsedNodes := NumUsedNodes - 1;
                            Shape1.Brush.Color := clFuchsia;
                            Shape1.Brush.Style := bsSolid;
                            Main(Sender)
                        end;
                InputPanel := Distance
            end
end;

procedure TWeb_Spinner.MovePanel(var Box: TPanel; X, Y: Integer);
begin
    Box.Left := Box.Left + (X - MouseOrigin.X);
    Box.Top := Box.Top + (Y - MouseOrigin.Y)
end;

procedure TWeb_Spinner.MoveBitBtn(var Node: TBitBtn; X, Y: Integer);
begin
    Node.Left := Node.Left + (X - MouseOrigin.X);
    Node.Top := Node.Top + (Y - MouseOrigin.Y)
end;

procedure TWeb_Spinner.GenericMouseMove(Sender: TObject; Shift:
TShiftState;
    X, Y: Integer);
var Loop: integer;
begin
    if Moving then begin

        if ((Sender = Panel1) or (Sender = Label4)) then
            MovePanel(Panel1, X, Y)
        else if (Sender = Panel2) then MovePanel(Panel2, X, Y)
        else if ((Sender = Panel14) or (Sender = Label19)
            or (Sender = Shape2)) then
            MovePanel(Panel14, X, Y)
        else if ((Sender = Panel18) or (Sender = Panel23))
            then MovePanel(Panel18, X, Y)
        else if ((Sender = Panel24) or (Sender = Panel25))
            then MovePanel(Panel24, X, Y)
        else if ((Sender = Panel28) or (Sender = Panel29))
```

```
                 then MovePanel(Panel28, X, Y);


     for Loop := 1 to NumNodes do
        if Sender = Nodes[Loop] then MoveBitBtn(Nodes[loop], X, Y)


  end
end;


procedure TWeb_Spinner.GenericMouseUp(Sender: TObject; Button:
TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  loop: integer;
begin
  if Sender = Panel1 then begin
    for loop := 1 to NumNodes do begin
       if Nodes[loop].Visible = false then begin
          Nodes[loop].Top := Panel1.Top + 24;
          Nodes[loop].Left := Panel1.Left + 24
       end
    end
  end
  else Main(Sender);

  if Moving then Moving := false
end;


{******************************************************************
 **   This procedure initializes the network matrix that defines      **
 **   the network structure.  It is executed everytime a new network  **
 **   is designed.                                                    **
 ******************************************************************}

procedure TWeb_Spinner.InitializeLinks(var Connections: LinkMatrixType);
var
  Row, Column: Integer;
begin
  for Row := 1 to NumNodes do
     for Column := 1 to NumNodes do
        if Connections[Row, Column] <> LockLinked then
           Connections[Row, Column] := NoLink
end;


{******************************************************************
 **   This procedure allows the user to "lock" a link into place in   **
 **   proposed network.                                               **
 ******************************************************************}

procedure TWeb_Spinner.LockTheLink(Sender: TObject; Button: TMouseButton;
                                   Shift: TShiftState; X, Y: Integer);
var
  Row, Column: Integer;
  NodesLinked: String;
  Node1, Node2: Integer;
```

```
begin
   if Button = mbLeft then begin
      for Row := 1 to NumUsedNodes do begin
         if Sender = Links[Row] then begin
            if Links[Row].Pen.Color = clFuchsia then begin
               if MessageDlg('LOCK ' + Links[Row].Hint + '?', mtConfirmation,
               mbOkCancel, 0) = mrOk then
               begin
                  Links[Row].Pen.Color := clBlack;
                  NodesLinked := Links[Row].Hint;
                  Node1 := StrToInt(NodesLinked[1]) * 10
                           + StrToInt(NodesLinked[2]);
                  Node2 := StrToInt(NodesLinked[4]) * 10
                           + StrToInt(NodesLinked[5]);
                  Connections[Node1, Node2] := LockLinked;
                  Connections[Node2, Node1] := LockLinked
               end
            end
            else if Links[Row].Pen.Color = clBlack then begin
               if MessageDlg('UNLOCK ' + Links[Row].Hint + '?',
                  mtConfirmation,
               mbOkCancel, 0) = mrOk then
               begin
                  Links[Row].Pen.Color := clFuchsia;
                  NodesLinked := Links[Row].Hint;
                  Node1 := StrToInt(NodesLinked[1]) * 10
                           + StrToInt(NodesLinked[2]);
                  Node2 := StrToInt(NodesLinked[4]) * 10
                           + StrToInt(NodesLinked[5]);
                  Connections[Node1, Node2] := Linked;
                  Connections[Node2, Node1] := Linked
               end
            end
         end
      end
   end;
   Main(Sender)
end;


{*********************************************************************
 **   This is a generic routine that is used to initialize all or    **
 **   part of any data matrix that it receives.                      **
 *********************************************************************}

procedure TWeb_Spinner.InitializeMatrix(var DataMatrix: MasterMatrixType;
                                        Data : Integer; Start:
Integer);
var
   Row, Column: Integer;
begin
   for Row := Start to NumNodes do
      for Column := 1 to NumNodes do
         DataMatrix[Row, Column] := Data;

   if Start > 1 then
      for Row := 1 to Start do
```

```
        for Column := Start to NumNodes do
          DataMatrix[Row, Column] := Data
end;



{**********************************************************************
  **   This procedure is used to fill the input matrix on the link    **
  **   characteristics screen with the data from any data matrix.      **
  **********************************************************************}

procedure TWeb_Spinner.FillInputMatrix(DataMatrix: MasterMatrixType;      .
                                       var InputMatrix: InputMatrixType);
var
  Row, Column: Integer;
begin

  for Row := 1 to NumNodes do
    for Column := 1 to NumNodes do begin
      InputMatrix[Row, Column].Color := clBlack;
      InputMatrix[Row, Column].Text := '      --';
      InputMatrix[Row, Column].ShowHint := false
    end;

  for Row := 1 to NumUsedNodes do
    for Column := 1 to NumUsedNodes do begin
      if Row > Column then InputMatrix[Row, Column].Color := clWhite
      else InputMatrix[Row, Column].Color := clGray;
    end;

  for Row := 1 to NumUsedNodes do
    for Column := 1 to NumUsedNodes do begin
      InputMatrix[Row, Column].ShowHint := true;
      if Row <> Column then
          InputMatrix[Row, Column].Text := IntToStr(DataMatrix[Row,
Column])
      else
          InputMatrix[Row, Column].Text := '      --'
    end

end;



{**********************************************************************
  **   Computes Pathagorean distance between any two points.           **
  **********************************************************************}

function TWeb_Spinner.DistanceBetweenPoints(Point1, Point2: TPoint):
Longint;
var
  Side1, Side2, Length1, Length2: LongInt;
  Answer: Real;
begin
  Side1 := Abs(Point1.X - Point2.X);
  Side2 := Abs(Point1.Y - Point2.Y);
  Length1 := Sqr(Side1);
  Length2 := Sqr(Side2);
  Answer := Sqrt(Abs(Length1 + Length2));
```

```
      Result := Trunc(Answer)
end;


{*********************************************************************
 **  Computes screen distance between nodes and stores this data in  **
 **  the Web Spinner Distance matrix.                                **
 ********************************************************************}

procedure TWeb_Spinner.DistanceIsCost(var DistanceMatrix:
MasterMatrixType;
                                       Nodes: NodeArrayType);
var
  CurrentNode, OtherNode: Integer;
  CtrPoint1, CtrPoint2: TPoint;
begin
  for CurrentNode := 1 to NumNodes do
    for OtherNode:= 1 to NumNodes do
      if CurrentNode < OtherNode then begin
        if (Nodes[CurrentNode].Visible and Nodes[OtherNode].Visible) then
        begin
          CtrPoint1 := Point(Nodes[CurrentNode].Left,
                             Nodes[CurrentNode].Top);
          CtrPoint2 := Point(Nodes[OtherNode].Left,
                             Nodes[OtherNode].Top);
          DistanceMatrix[CurrentNode, OtherNode] :=
                DistanceBetweenPoints(CtrPoint1, CtrPoint2);
          DistanceMatrix[OtherNode, CurrentNode] :=
                DistanceMatrix[CurrentNode, OtherNode]
        end
      end
end;


{*********************************************************************
 **  Extracts user-input data from the Link Characteristics screen and  **
 **  stores this data in the appropriate data matrix.                    **
 ********************************************************************}

procedure TWeb_Spinner.ExtractInputs(var DataMatrix: MasterMatrixType;
                                      InputMatrix: InputMatrixType);
var
  Row, Column, Warning: Integer;
  Notice: String;
begin
  Warning := 1;
  for Row := 1 to NumUsedNodes do
    for Column := 1 to NumUsedNodes do
      if Row < Column then begin
        try
          Warning := StrToIntDef(Inputmatrix[Row, Column].Text, -1);
        except
          begin
            Notice := 'Value for Link joining Nodes ' + IntToStr(Column)
                      + '/' + IntToStr(Row) + ' is not a valid Integer
                      value.';
            MessageDlg(Notice, mtError, [mbOK], 0);
```

```
                    Notebook1.PageIndex := 1
                  end
                end;
                  DataMatrix[Row, Column] := Warning;
                  DataMatrix[Column, Row] := DataMatrix[Row, Column]
            end
end;


{***********************************************************************
  **   Interpolates utility value for input values that fall between   **
  **   data points.                                                    **
  ***********************************************************************}

function TWeb_Spinner.Interpolate(Input, XAxisLeft, XAxisRight,
                                    ValueLeft, ValueRight: Integer): Real;
var
  AxisDiff, ValueDiff : Integer;
  HowMuch, FinalValue : Real;
begin
  AxisDiff := XAxisRight - XAxisLeft;
  ValueDiff := ValueRight - ValueLeft;

  HowMuch := (Input - XAxisLeft)/AxisDiff;
  FinalValue := ValueLeft + (HowMuch * ValueDiff);

  Result := FinalValue
end;


{***********************************************************************
  **   Retrieves appropriate utility value from the respective         **
  **   utility chart.                                                  **
  ***********************************************************************}

function TWeb_Spinner.GetUtility(Input: Integer;
                                    InputType: InputPanelType): Integer;
var
  UtilityValue: Real;
  Loop, Left, Right: Integer;

begin
  if InputType = SetUpCost then begin
    if Input <= UtilSetUpCostXAxis[1]
      then UtilityValue := UtilSetUpCostArray[1]
    else if Input >= UtilSetUpCostXAxis[10]
      then UtilityValue := UtilSetUpCostArray[10]
    else begin
      for Loop := 1 to 9 do begin
        if ((Input >= UtilSetUpCostXAxis[Loop]) and
            (Input < UtilSetUpCostXAxis[Loop + 1])) then begin
          Left := Loop;
          Right := Loop + 1
        end
      end;
      UtilityValue := Interpolate(Input, UtilSetUpCostXAxis[Left],
                                    UtilSetUpCostXAxis[Right],
```

```
                                UtilSetUpCostArray[Left],
                                UtilSetUpCostArray[Right])
      end
   end

   else if InputType = MonthlyCost then begin
    if Input <= UtilMonthlyCostXAxis[1]
       then UtilityValue := UtilMonthlyCostArray[1]
      else if Input >= UtilMonthlyCostXAxis[10]
       then UtilityValue := UtilMonthlyCostArray[10]
      else begin
        for Loop := 1 to 9 do begin
           if ((Input >= UtilMonthlyCostXAxis[Loop]) and
              (Input < UtilMonthlyCostXAxis[Loop + 1])) then begin
             Left := Loop;
             Right := Loop + 1
           end
         end;
         UtilityValue := Interpolate(Input, UtilMonthlyCostXAxis[Left],
                                    UtilMonthlyCostXAxis[Right],
                                    UtilMonthlyCostArray[Left],
                                    UtilMonthlyCostArray[Right])
      end
   end

   else if InputType = BER then begin
     if Input <= UtilBERXAxis[1]
       then UtilityValue := UtilBERArray[1]
      else if Input >= UtilBERXAxis[10]
       then UtilityValue := UtilBERArray[10]
      else begin
        for Loop := 1 to 9 do begin
           if ((Input >= UtilBERXAxis[Loop]) and
              (Input < UtilBERXAxis[Loop + 1])) then begin
             Left := Loop;
             Right := Loop + 1
           end
         end;
         UtilityValue := Interpolate(Input, UtilBERXAxis[Left],
                                    UtilBERXAxis[Right],
                                    UtilBERArray[Left],
                                    UtilBERArray[Right])
      end
   end;

   Result := Trunc(UtilityValue)
end;


{**********************************************************************
 **  Computes total utility values for each potential link and stores  **
 **  this information in the Web Spinner Utility Matrix.                **
 **********************************************************************}

procedure TWeb_Spinner.FillUtilityMatrix(var UtilityMatrix:
                                        MasterMatrixType;
```

87

```
                                      SetUpCostMatrix,
                                      MonthlyCostMatrix,
                                      BERMatrix: MasterMatrixType;
                                      SetUpCostWt, MonthlyCostWt,
                                      BERWt: Integer);
  var
    Row, Column: Integer;
    SetUpCostUtil, MonthlyCostUtil, BERUtil: Integer;
  begin
    for Row := 1 to NumUsedNodes do
      for Column := 1 to NumUsedNodes do
        if Row > Column then begin
          SetUpCostUtil := 0;
          MonthlyCostUtil := 0;
          BERUtil := 0;

          if CheckBox1.Checked then
            SetUpCostUtil := GetUtility(SetUpCostMatrix[Row, Column],
                                        SetUpCost);
          if CheckBox2.Checked then
            MonthlyCostUtil := GetUtility(MonthlyCostMatrix[Row, Column],
                                          MonthlyCost);
          if CheckBox3.Checked then
            BERUtil := GetUtility(BERMatrix[Row, Column], BER);

          UtilityMatrix[Row, Column] := ((SetUpCostUtil * SetUpCostWt) +
                                         (MonthlyCostUtil * MonthlyCostWt) +
                                         (BERUtil * BERWt)) div 100;
          UtilityMatrix[Column, Row] := UtilityMatrix[Row, Column]
        end
  end;


{**********************************************************************
 **   Creates a Minimum Spanning Tree.                              **
 **********************************************************************}

procedure TWeb_Spinner.SolveMinTree(var DataMatrix: MasterMatrixType;
                                     ExpenseMatrix: MasterMatrixType;
                                 var Connections: LinkMatrixType);
var
  EligibleParents : Array[1..NumNodes] of Integer;
  AdoptiveChildren : Array[1..NumNodes] of Integer;
  Loop, PCount, CCount: Integer;
  Parent, Child, Util: Integer;
  Locked: Boolean;
begin
  for Loop := 1 to NumNodes do begin
    EligibleParents[Loop] := 0;
    AdoptiveChildren[Loop] := Loop
  end;
  EligibleParents[1] := 1;
  AdoptiveChildren[1] := 0;
  Locked := false;

  for Loop := 1 to NumUsedNodes do begin
    Parent := 0;
```

88

```
      Child := 0;
      Util := Infinity;

      for PCount := 1 to NumUsedNodes do begin
        if EligibleParents[PCount] <> 0 then begin

          for CCount := 1 to NumUsedNodes do begin
            if AdoptiveChildren[CCount] <> 0 then begin
              if Connections[PCount, CCount] = LockLinked then begin
                Util := -1;
                Parent := PCount;
                Child := CCount;
                Locked := true
              end;
              if DataMatrix[PCount, CCount] < Util then begin
                Util := DataMatrix[PCount, CCount];
                Parent := PCount;
                Child := CCount
              end
            end
          end
        end
      end;

    If ((Util < Infinity) and (Parent > 0) and (Child > 0)) then begin
      EligibleParents[Child] := Child;
      AdoptiveChildren[Child] := 0;

      if not Locked then begin
        Connections[Parent, Child] := Linked;
        Connections[Child, Parent] := Linked
      end;

      Locked := false
    end
  end
end;


{*******************************************************************
  **   Creates a Maximum Spanning Tree.                          **
  *******************************************************************}

procedure TWeb_Spinner.SolveMaxTree(var DataMatrix: MasterMatrixType;
                                    ExpenseMatrix: MasterMatrixType;
                                    var Connections: LinkMatrixType);
var
  EligibleParents : Array[1..NumNodes] of Integer;
  AdoptiveChildren : Array[1..NumNodes] of Integer;
  Loop, PCount, CCount: Integer;
  Parent, Child, Util: Integer;
  Locked: Boolean;
begin
  for Loop := 1 to NumNodes do begin
    EligibleParents[Loop] := 0;
    AdoptiveChildren[Loop] := Loop
  end;
```

```
      EligibleParents[1] := 1;
      AdoptiveChildren[1] := 0;
      Locked := false;

      for Loop := 1 to NumUsedNodes do begin
        Parent := 0;
        Child := 0;
        Util := 0;

        for PCount := 1 to NumUsedNodes do begin
          if EligibleParents[PCount] <> 0 then begin

            for CCount := 1 to NumUsedNodes do begin
              if AdoptiveChildren[CCount] <> 0 then begin
                if Connections[PCount, CCount] = LockLinked then begin
                  Util := Infinity - 1;
                  Parent := PCount;
                  Child := CCount;
                  Locked := true
                end;
                if DataMatrix[PCount, CCount] > Util then begin
                  Util := DataMatrix[PCount, CCount];
                  Parent := PCount;
                  Child := CCount
                end
              end
            end
          end
        end;

        If ((Util < Infinity) and (Parent > 0) and (Child > 0)) then begin
          EligibleParents[Child] := Child;
          AdoptiveChildren[Child] := 0;

          if not Locked then begin
            Connections[Parent, Child] := Linked;
            Connections[Child, Parent] := Linked
          end;

          Locked := false
        end
      end
    end;


{********************************************************************
 **   This routine checks the connection matrix and draws the      **
 **   appropriate links on the Web Spinner desktop.                **
 ********************************************************************}

procedure TWeb_Spinner.DrawLinks(Connections: LinkMatrixType;
                                 Nodes: NodeArrayType; var Links:
LinkArrayType);
var
  Loop, Row, Column: Integer;
begin
  for Loop := 1 to NumNodes do begin
```

```
      Links[Loop].Pen.Color := clFuchsia;
      Links[Loop].Visible := false
   end;

   Loop := 1;
   for Row := 1 to NumNodes do
      for Column := 1 to NumNodes do
         if Row <= Column then begin
            if Connections[Row, Column] = LockLinked then
               Links[Loop].Pen.Color := clBlack
            else if Connections[Row, Column] = Linked then
               Links[Loop].Pen.Color := clFuchsia
            else if Connections[Row, Column] = Redundant then
               Links[Loop].Pen.Color := clLime;

            if ((Connections[Row, Column] = Linked) or
                (Connections[Row, Column] = LockLinked) or
                (Connections[Row, Column] = Redundant)) then begin

               if Nodes[Row].Top < Nodes[Column].Top then begin
                  if Nodes[Row].Left < Nodes[Column].Left then begin
                     Links[Loop].Top := Nodes[Row].Top + 12;
                     Links[Loop].Left := Nodes[Row].Left + 12;
                     Links[Loop].Height := Abs(Nodes[Row].Top
                                                - Nodes[Column].Top);
                     Links[Loop].Width := Abs(Nodes[Row].Left
                                                - Nodes[Column].Left);
                     Links[Loop].Shape := stDiagLine1
                  end
                  else if Nodes[Row].Left > Nodes[Column].Left then begin
                     Links[Loop].Top := Nodes[Row].Top + 12;
                     Links[Loop].Left := Nodes[Column].Left + 12;
                     Links[Loop].Height := Abs(Nodes[Row].Top
                                                - Nodes[Column].Top);
                     Links[Loop].Width := Abs(Nodes[Row].Left
                                                - Nodes[Column].Left);
                     Links[Loop].Shape := stDiagLine2
                  end
                  else if Nodes[Row].Left = Nodes[Column].Left then begin
                     Links[Loop].Top := Nodes[Row].Top + 12;
                     Links[Loop].Left := Nodes[Row].Left + 12;
                     Links[Loop].Height := Abs(Nodes[Row].Top
                                                - Nodes[Column].Top);
                     Links[Loop].Width := 5;
                     Links[Loop].Shape := stLeftLine
                  end
               end

               else if Nodes[Row].Top > Nodes[Column].Top then begin
                  if Nodes[Row].Left < Nodes[Column].Left then begin
                     Links[Loop].Top := Nodes[Column].Top + 12;
                     Links[Loop].Left := Nodes[Row].Left + 12;
                     Links[Loop].Height := Abs(Nodes[Row].Top
                                                - Nodes[Column].Top);
                     Links[Loop].Width := Abs(Nodes[Row].Left
                                                - Nodes[Column].Left);
                     Links[Loop].Shape := stDiagLine2
```

```
      end
    else if Nodes[Row].Left > Nodes[Column].Left then begin
      Links[Loop].Top := Nodes[Column].Top + 12;
      Links[Loop].Left := Nodes[Column].Left + 12;
      Links[Loop].Height := Abs(Nodes[Row].Top
                                    - Nodes[Column].Top);
      Links[Loop].Width := Abs(Nodes[Row].Left
                                    - Nodes[Column].Left);
      Links[Loop].Shape := stDiagLine1
    end
    else if Nodes[Row].Left = Nodes[Column].Left then begin
      Links[Loop].Top := Nodes[Column].Top + 12;
      Links[Loop].Left := Nodes[Column].Left + 12;
      Links[Loop].Height := Abs(Nodes[Row].Top
                                    - Nodes[Column].Top);
      Links[Loop].Width := 5;
      Links[Loop].Shape := stLeftLine
    end
  end

  else if Nodes[Row].Top = Nodes[Column].Top then begin
    if Nodes[Row].Left < Nodes[Column].Left then begin
      Links[Loop].Top := Nodes[Row].Top + 12;
      Links[Loop].Left := Nodes[Row].Left + 12;
      Links[Loop].Height := 5;
      Links[Loop].Width := Abs(Nodes[Row].Left
                                    - Nodes[Column].Left);
      Links[Loop].Shape := stTopLine
    end
    else if Nodes[Row].Left > Nodes[Column].Left then begin
      Links[Loop].Top := Nodes[Row].Top + 12;
      Links[Loop].Left := Nodes[Column].Left + 12;
      Links[Loop].Height := 5;
      Links[Loop].Width := Abs(Nodes[Row].Left
                                    - Nodes[Column].Left);
      Links[Loop].Shape := stTopLine
    end
    else if Nodes[Row].Left = Nodes[Column].Left then begin
      Links[Loop].Top := Nodes[Row].Top + 12;
      Links[Loop].Left := Nodes[Row].Left + 12;
      Links[Loop].Height := 5;
      Links[Loop].Width := 5;
      Links[Loop].Shape := stTopLine
    end
  end;

  Links[Loop].Visible := true;
  if Row < 10 then Links[Loop].Hint := '0' + IntToStr(Row) + '.'
  else Links[Loop].Hint := IntToStr(Row) + '.';

  if Column < 10 then
    Links[Loop].Hint := Links[Loop].Hint + '0' + IntToStr(Column)
  else Links[Loop].Hint := Links[Loop].Hint + IntToStr(Column);

  ListBox1.Items.Add(InttoStr(Row) + ',' + IntToStr(Column));
  Loop := Loop + 1
end
```

```
        end
end;


{*******************************************************************
 **   Computes the sum of a particular attribute of the recommended  **
 **   network (i.e. total set-up cost, total utility,...).           **
 *****************************************************************}

function TWeb_Spinner.ComputeTotal(DataMatrix: MasterMatrixType;
                                    Connections: LinkMatrixType): Integer;
var
  Row, Col, Value: Integer;
begin
  Value := 0;
  for Row := 1 to NumUsedNodes do
    for Col := 1 to NumUsedNodes do
      if Row > Col then begin
        if Connections[Row, Col] <> NoLink
          then Value := Value + DataMatrix[Row, Col]
      end;
  Result := Value
end;


{*******************************************************************
 **   Main WebSpinner Routine.  Executed every time any modification **
 **   is made to the network or any solution data. Responsible for    **
 **   central Web Spinner program execution with the assistance of    **
 **   all other methods.  All methods support this routine.           **
 *****************************************************************}

procedure TWeb_Spinner.Main(Sender: TObject);
begin
  InitializeLinks(Connections);
  ListBox1.Items.Clear;

  if CheckBox4.Checked then begin
    InitializeMatrix(DistanceMatrix, Infinity, 1);
    DistanceIsCost(DistanceMatrix, Nodes);
    SolveMinTree(DistanceMatrix, DistanceMatrix, Connections);
    TotalCost := ComputeTotal(DistanceMatrix, Connections);
    TotalUtil := TotalCost
  end
  else begin
    InitializeMatrix(UtilityMatrix, 0, 1);
    InitializeMatrix(SetUpCostMatrix, 0, NumUsedNodes + 1);
    InitializeMatrix(MonthlyCostMatrix, 0, NumUsedNodes + 1);
    InitializeMatrix(BERMatrix, 0, NumUsedNodes + 1);

    SetUpCostWt := SpinEdit1.Value;
    MonthlyCostWt := SpinEdit3.Value;
    BERWt := SpinEdit2.Value;

    FillUtilityMatrix(UtilityMatrix, SetUpCostMatrix, MonthlyCostMatrix,
                       BERMatrix, SetUpCostWt, MonthlyCostWt, BERWt);
```

93

```
      SolveMaxTree(UtilityMatrix, SetUpCostMatrix, Connections);
      TotalCost := ComputeTotal(SetUpCostMatrix, Connections);
      TotalUtil := ComputeTotal(UtilityMatrix, Connections)
    end;

  DrawLinks(Connections, Nodes, Links);
  Edit1.Text := IntToStr(TotalCost);
  Edit8.Text := IntToStr(TotalUtil);
  Edit3.Text := IntToStr(NumUsedNodes)
end;


{*******************************************************************
 **   Generates a new node on the desktop.                      **
 *******************************************************************}

procedure TWeb_Spinner.Shape1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var
  AvailableNode : Boolean;
  Loop : Integer;
begin
  AvailableNode := true;

  for Loop := 1 to NumNodes do begin
    if (Nodes[Loop].Visible = false) and AvailableNode then begin
      Nodes[Loop].Visible := true;
      NumUsedNodes := NumUsedNodes + 1;
      AvailableNode := false
    end
    else if (Loop = NumNodes) and AvailableNode then begin
            Shape1.Brush.Color := clRed;
            Shape1.Brush.Style := bsDiagCross
    end
  end
end;


{*******************************************************************
 **   Prints the currently displayed screen.                    **
 *******************************************************************}

procedure TWeb_Spinner.Print1Click(Sender: TObject);
begin
  TabSet1.Visible := false;
  Panel1.Visible := false;
  Print;
  TabSet1.Visible := true;
  Panel1.Visible := true
end;


{*******************************************************************
 **   Returns program control to the WSAbout form when activated. **
 **   Displays the initial welcome page.                         **
 *******************************************************************}
```

```
procedure TWeb_Spinner.About1Click(Sender: TObject);
begin
  About.Show;
end;


{*******************************************************************
 **   Activates the Set-Up cost matrix on the Data Input Screen.    **
 ******************************************************************}

procedure TWeb_Spinner.SpeedButton1Click(Sender: TObject);
begin
  OnExitInputBox(Sender);
  if InputPanel = Traffic then ExtractInputs(TrafficMatrix, InputMatrix)
  else if InputPanel = SetUpCost then
ExtractInputs(SetUpCostMatrix,InputMatrix)
  else if InputPanel = MonthlyCost then ExtractInputs(MonthlyCostMatrix,
                                                 InputMatrix)
  else if InputPanel = BER then ExtractInputs(BERMatrix, InputMatrix);

  Label1.Caption := 'SetUp Cost';
  Label1.Color := clTeal;
  Panel8.Color := clTeal;
  ListBox2.Items.Clear;
  ListBox2.Items.Add('Initial SetUp cost to connect');
  ListBox2.Items.Add('nodes. (Max = $' + IntToStr(Infinity - 1) + ')');
  InputPanel := SetUpCost;
  FillInputMatrix(SetUpCostMatrix, InputMatrix)
end;


{*******************************************************************
 **   Activates the BER matrix on the Data Input Screen.            **
 ******************************************************************}

procedure TWeb_Spinner.SpeedButton4Click(Sender: TObject);
begin
  OnExitInputBox(Sender);
  if InputPanel = Traffic then ExtractInputs(TrafficMatrix, InputMatrix)
  else if InputPanel = SetUpCost then
      ExtractInputs(SetUpCostMatrix,InputMatrix)
  else if InputPanel = MonthlyCost then ExtractInputs(MonthlyCostMatrix,
                                                 InputMatrix)
  else if InputPanel = BER then ExtractInputs(BERMatrix, InputMatrix);

  Label1.Caption := 'Error Rate';
  Label1.Color := clGreen;
  Panel8.Color := clGreen;
  ListBox2.Items.Clear;
  ListBox2.Items.Add('Error Rate of each Link.');
  ListBox2.Items.Add('(10E-#)(negative values)');
  InputPanel := BER;
  FillInputMatrix(BERMatrix, InputMatrix)

end;
```

```
{************************************************************************
 **   Activates the Monthly cost matrix on the Data Input Screen.     **
 ***********************************************************************}

procedure TWeb_Spinner.SpeedButton2Click(Sender: TObject);
begin
  OnExitInputBox(Sender);
  if InputPanel = Traffic then ExtractInputs(TrafficMatrix, InputMatrix)
  else if InputPanel = SetUpCost then
       ExtractInputs(SetUpCostMatrix,InputMatrix)
  else if InputPanel = MonthlyCost then ExtractInputs(MonthlyCostMatrix,
                                                      InputMatrix)
  else if InputPanel = BER then ExtractInputs(BERMatrix, InputMatrix);

  Label1.Caption := 'Monthly Cost';
  Label1.Color := clNavy;
  Panel8.Color := clNavy;
  ListBox2.Items.Clear;
  ListBox2.Items.Add('Monthly Costs to maintain');
  ListBox2.Items.Add('each link. (Max = $' + IntToStr(Infinity - 1)
                     + ')');
  InputPanel := MonthlyCost;
  FillInputMatrix(MonthlyCostMatrix, InputMatrix)
end;


{************************************************************************
 ** Activates the Traffic Requirements matrix on the Data Input Screen. **
 ***********************************************************************}

procedure TWeb_Spinner.SpeedButton3Click(Sender: TObject);
begin
  OnExitInputBox(Sender);
  if InputPanel = Traffic then ExtractInputs(TrafficMatrix, InputMatrix)
  else if InputPanel = SetUpCost then
       ExtractInputs(SetUpCostMatrix,InputMatrix)
  else if InputPanel = MonthlyCost then ExtractInputs(MonthlyCostMatrix,
                                                      InputMatrix)
  else if InputPanel = BER then ExtractInputs(BERMatrix, InputMatrix);

  Label1.Caption := 'Traffic';
  Label1.Color := clAqua;
  Panel8.Color := clAqua;
  ListBox2.Items.Clear;
  ListBox2.Items.Add('Traffic Requirements ');
  ListBox2.Items.Add('between nodes (packets).');
  InputPanel := Traffic;
  FillInputMatrix(TrafficMatrix, InputMatrix)
end;


{************************************************************************
 **   Ensures all matrix data is acknowledged when user changes       **
 **   between Web Spinner modules.                                    **
 ***********************************************************************}

procedure TWeb_Spinner.Notebook1PageChanged(Sender: TObject);
```

```
begin
  if NumUsedNodes > 0 then begin
    if InputPanel = SetUpCost then ExtractInputs(SetUpCostMatrix,
                                                 InputMatrix)
    else if InputPanel = MonthlyCost then
      ExtractInputs(MonthlyCostMatrix,
                                          InputMatrix)
    else if InputPanel = BER then ExtractInputs(BERMatrix, InputMatrix)
    else if InputPanel = Traffic then ExtractInputs(TrafficMatrix,
                                                    InputMatrix);
    if NumUsedNodes > 0 then Main(Sender)
  end
end;


{******************************************************************
 **   Fills in the upper right half of data input matrix every time  **
 **   the user enter data on the data input screen.                  **
 ******************************************************************}

procedure TWeb_Spinner.OnExitInputBox(Sender: TObject);
var Row, Column: Integer;
begin
  for Row := 1 to NumUsedNodes do
    for Column := 1 to NumUsedNodes do
      if Row > Column then
        InputMatrix[Column, Row].Text := InputMatrix[Row, Column].Text
      else if Row > Column then InputMatrix[Row, Column].Text := '0'

end;


{******************************************************************
 **   Displays the Utility matrix on the Data Input Screen.          **
 ******************************************************************}

procedure TWeb_Spinner.Button1Click(Sender: TObject);
begin
  OnExitInputBox(Sender);
  if InputPanel = Traffic then ExtractInputs(TrafficMatrix, InputMatrix)
  else if InputPanel = SetUpCost then
      ExtractInputs(SetUpCostMatrix,InputMatrix)
  else if InputPanel = MonthlyCost then ExtractInputs(MonthlyCostMatrix,
                                                      InputMatrix)
  else if InputPanel = BER then ExtractInputs(BERMatrix, InputMatrix);

  InputPanel := Utility;
  Label1.Caption := 'Utility';
  Label1.Color := clWhite;
  Panel8.Color := clWhite;
  ListBox2.Items.Clear;
  ListBox2.Items.Add('Current Utility Values ');
  ListBox2.Items.Add('between node (read only).');
  FillInputMatrix(UtilityMatrix, InputMatrix);
  Main(Sender)
end;
```

```
{**********************************************************************
**   Computes weight total every time user makes a change to the    **
**   current weighting distribution.   (Weights poge.)              **
**********************************************************************}

procedure TWeb_Spinner.TotalWeights(Sender: TObject);
begin
  try
    Edit9.Text := IntToStr(SpinEdit1.Value + SpinEdit2.Value +
SpinEdit3.Value);
  except
    on EConvertError do begin
      SpinEdit1.Value := 0;
      SpinEdit2.Value := 0;
      SpinEdit3.Value := 0;
      Edit9.Text := IntToStr(SpinEdit1.Value + SpinEdit2.Value +
                             SpinEdit3.Value);
    end;
  end;

  ChartFX1.OpenData[COD_VALUES] := MakeLong(1,3);

  ChartFX1.ThisSerie := 0;
  ChartFX1.Value[0] := SpinEdit1.Value;
  ChartFX1.Value[1] := SpinEdit2.Value;
  ChartFX1.Value[2] := SpinEdit3.Value;

  ChartFX1.CloseData[COD_VALUES] := 0
end;


{**********************************************************************
**   The following procedures are used to hide and/or unhide tools  **
**   and panels during program execution.                           **
**********************************************************************}

procedure TWeb_Spinner.BitBtn2Click(Sender: TObject);
begin
  Panel2.Visible := false
end;

procedure TWeb_Spinner.Tools1Click(Sender: TObject);
begin
  Panel14.Visible := true
end;

procedure TWeb_Spinner.BitBtn5Click(Sender: TObject);
begin
  Panel14.Visible := false;
  Edit10.Text := '';
  Edit11.Text := ''
end;

procedure TWeb_Spinner.LinkInfoBox1Click(Sender: TObject);
begin
  if not Panel2.Visible then Panel2.Visible := true
```

```
    else Panel2.Visible := false
end;


procedure TWeb_Spinner.BitBtn8Click(Sender: TObject);
begin
   Panel18.Visible := false
end;


procedure TWeb_Spinner.BitBtn10Click(Sender: TObject);
begin
    Panel24.Visible := false
end;


procedure TWeb_Spinner.Solve1Click(Sender: TObject);
begin
   if not Panel24.Visible then Panel24.Visible := true
   else Panel24.Visible := false
end;


procedure TWeb_Spinner.BasicData2Click(Sender: TObject);
begin
   ComboBox1.Text := IntToStr(LinkSpeed);
   SpinEdit4.Value := PacketSize;
   if not Panel18.Visible then Panel18.Visible := true
   else Panel18.Visible := false
end;


procedure TWeb_Spinner.BitBtn12Click(Sender: TObject);
begin
   Panel28.Visible := false
end;


procedure TWeb_Spinner.LinkLegend1Click(Sender: TObject);
begin
   if not Panel28.Visible then Panel28.Visible := true
   else Panel28.Visible := false
end;



{******************************************************************
**   This is the "Ok" button on the Lock Link tool.  When executed,   **
**   creates a locked link between indicated nodes.                    **
******************************************************************}

procedure TWeb_Spinner.BitBtn4Click(Sender: TObject);
var
   LinkStart, LinkEnd: Integer;
begin
   LinkStart := StrToIntDef(Edit10.Text, 0);
   LinkEnd := StrToIntDef(Edit11.Text, 0);

   if ((LinkStart = 0) or (LinkEnd = 0)) then
     MessageDlg('Nodes must be Integers > 0.', mtError, [mbOk], 0);

   if ((LinkStart <> 0) and (LinkEnd <> 0) and (LinkStart = LinkEnd)) then
     MessageDlg('Cannot create a link starting and ending at the same
                 node.', mtError, [mbOk], 0);
```

99

```
    if LinkStart = 0 then Edit10.Text := '';
    if LinkEnd = 0 then Edit11.Text := '';

    if ((LinkStart > 0) and (LinkEnd > 0) and (LinkStart <> LinkEnd)) then
    begin
      Connections[LinkStart, LinkEnd] := LockLinked;
      Connections[LinkEnd, LinkStart] := LockLinked;
      Main(Sender);
      Panel14.Visible := false;
      Edit10.Text := '';
      Edit11.Text := ''
    end
end;



{***********************************************************************
 **   Switches data points between any two charts.                    **
 **********************************************************************}

procedure TWeb_Spinner.SwitchGraphValues(var UtilArray: UtilityGraphType;
            var SmallChart: TChartFX; MainChart: TChartFX);
var
  Loop: Integer;
begin
  SmallChart.OpenData[COD_VALUES] := MakeLong(1,10);
  SmallChart.ThisSerie := 0;
  MainChart.ThisSerie := 0;

  for Loop := 0 to 9 do begin
    SmallChart.Value[Loop] := MainChart.Value[Loop];
    UtilArray[Loop + 1] := Trunc(MainChart.Value[Loop])
  end;

  SmallChart.CloseData[COD_VALUES] := 0
end;



{***********************************************************************
 ** The next three procedures are used whenever the user double-clicks **
 ** any of the three small charts on the Utility screen.  They          **
 ** activate the respective chart, transfer appropriate data points,    **
 ** and fill in the appropriate labels.                                 **
 **********************************************************************}

procedure TWeb_Spinner.ChartFX2LButtonDblClk(Sender: TObject; var X, Y,
  nSerie, nPoint, nRes: Integer);
var
  Loop: Integer;
begin
  if UtilityGraph = BER then SwitchGraphValues(UtilBERArray,
                                          ChartFX3, ChartFX6);
  if UtilityGraph = MonthlyCost then
      SwitchGraphValues(UtilMonthlyCostArray,ChartFX4, ChartFX6);

  UtilityGraph := SetUpCost;
  Panel19.Caption := 'SetUp Cost ($)';
```

100

```
    Panel19.Color := clTeal;
    Shape3.Brush.Color := clTeal;

    ChartFX2.RGB2DBk := clYellow;
    ChartFX3.RGB2DBk := clWhite;
    ChartFX4.RGB2DBk := clWhite;

    ChartFX2.OpenData[COD_VALUES] := MakeLong(1,10);
    ChartFX6.OpenData[COD_VALUES] := MakeLong(1,10);
    ChartFX2.ThisSerie := 0;
    ChartFX6.ThisSerie := 0;
    for Loop := 0 to 9 do begin
       ChartFX2.Value[Loop] := UtilSetUpCostArray[Loop + 1];
       ChartFX6.Value[Loop] := UtilSetUpCostArray[Loop + 1]
    end;
    ChartFX2.CloseData[COD_VALUES] := 0;
    ChartFX6.CloseData[COD_VALUES] := 0;

    Edit4.Text := IntToStr(UtilSetUpCostArray[1]);
    Edit5.Text := IntToStr(UtilSetUpCostArray[2]);
    Edit6.Text := IntToStr(UtilSetUpCostArray[3]);
    Edit7.Text := IntToStr(UtilSetUpCostArray[4]);
    Edit12.Text := IntToStr(UtilSetUpCostArray[5]);
    Edit13.Text := IntToStr(UtilSetUpCostArray[6]);
    Edit14.Text := IntToStr(UtilSetUpCostArray[7]);
    Edit15.Text := IntToStr(UtilSetUpCostArray[8]);
    Edit16.Text := IntToStr(UtilSetUpCostArray[9]);
    Edit17.Text := IntToStr(UtilSetUpCostArray[10]);

    for Loop := 0 to 9 do
       StringGrid1.Cells[Loop, 0] := IntToStr(UtilSetUpCostXAxis[Loop + 1])
end;

procedure TWeb_Spinner.ChartFX3LButtonDblClk(Sender: TObject; var X, Y,
   nSerie, nPoint, nRes: Integer);
var
   Loop: Integer;
begin
   if UtilityGraph = SetUpCost then SwitchGraphValues(UtilSetUpCostArray,
                                                      ChartFX2, ChartFX6);
   if UtilityGraph = MonthlyCost then
       SwitchGraphValues(UtilMonthlyCostArray, ChartFX4, ChartFX6);

   UtilityGraph := BER;
   Panel19.Caption := 'BER (10E-#)';
   Panel19.Color := clGreen;
   Shape3.Brush.Color := clGreen;

   ChartFX2.RGB2DBk := clWhite;
   ChartFX3.RGB2DBk := clYellow;
   ChartFX4.RGB2DBk := clWhite;

   ChartFX3.OpenData[COD_VALUES] := MakeLong(1,10);
   ChartFX6.OpenData[COD_VALUES] := MakeLong(1,10);
   ChartFX3.ThisSerie := 0;
   ChartFX6.ThisSerie := 0;
   for Loop := 0 to 9 do begin
```

```
      ChartFX3.Value[Loop]  := UtilBERArray[Loop + 1];
      ChartFX6.Value[Loop]  := UtilBERArray[Loop + 1]
    end;
  ChartFX3.CloseData[COD_VALUES]  := 0;
  ChartFX6.CloseData[COD_VALUES]  := 0;

  Edit4.Text  := IntToStr(UtilBERArray[1]);
  Edit5.Text  := IntToStr(UtilBERArray[2]);
  Edit6.Text  := IntToStr(UtilBERArray[3]);
  Edit7.Text  := IntToStr(UtilBERArray[4]);
  Edit12.Text := IntToStr(UtilBERArray[5]);
  Edit13.Text := IntToStr(UtilBERArray[6]);
  Edit14.Text := IntToStr(UtilBERArray[7]);
  Edit15.Text := IntToStr(UtilBERArray[8]);
  Edit16.Text := IntToStr(UtilBERArray[9]);
  Edit17.Text := IntToStr(UtilBERArray[10]);

  for Loop := 0 to 9 do
     StringGrid1.Cells[Loop, 0] := IntToStr(UtilBERXAxis[Loop + 1])
end;

procedure TWeb_Spinner.ChartFX4LButtonDblClk(Sender: TObject; var X, Y,
  nSerie, nPoint, nRes: Integer);
var
  Loop: Integer;
begin
  if UtilityGraph = SetUpCost then SwitchGraphValues(UtilSetUpCostArray,
                                             ChartFX2, ChartFX6);
  if UtilityGraph = BER then SwitchGraphValues(UtilBERArray,
                                             ChartFX3, ChartFX6);

  UtilityGraph := MonthlyCost;
  Panel19.Caption := 'Monthly Cost ($)';
  Panel19.Color := clNavy;
  Shape3.Brush.Color := clNavy;

  ChartFX2.RGB2DBk := clWhite;
  ChartFX3.RGB2DBk := clWhite;
  ChartFX4.RGB2DBk := clYellow;

  ChartFX4.OpenData[COD_VALUES] := MakeLong(1,10);
  ChartFX6.OpenData[COD_VALUES] := MakeLong(1,10);
  ChartFX4.ThisSerie := 0;
  ChartFX6.ThisSerie := 0;
  for Loop := 0 to 9 do begin
    ChartFX4.Value[Loop]  := UtilMonthlyCostArray[Loop + 1];
    ChartFX6.Value[Loop]  := UtilMonthlyCostArray[Loop + 1]
  end;
  ChartFX4.CloseData[COD_VALUES]  := 0;
  ChartFX6.CloseData[COD_VALUES]  := 0;

  Edit4.Text  := IntToStr(UtilMonthlyCostArray[1]);
  Edit5.Text  := IntToStr(UtilMonthlyCostArray[2]);
  Edit6.Text  := IntToStr(UtilMonthlyCostArray[3]);
  Edit7.Text  := IntToStr(UtilMonthlyCostArray[4]);
  Edit12.Text := IntToStr(UtilMonthlyCostArray[5]);
  Edit13.Text := IntToStr(UtilMonthlyCostArray[6]);
```

```
      Edit14.Text := IntToStr(UtilMonthlyCostArray[7]);
      Edit15.Text := IntToStr(UtilMonthlyCostArray[8]);
      Edit16.Text := IntToStr(UtilMonthlyCostArray[9]);
      Edit17.Text := IntToStr(UtilMonthlyCostArray[10]);

      for Loop := 0 to 9 do
        StringGrid1.Cells[Loop, 0] :=
          IntToStr(UtilMonthlyCostXAxis[Loop + 1])
  end;


{**********************************************************************
  **   The next four procedures either subtract (down) or add (up) one   **
  **   to the integer value in a textbox when the appropriate spin       **
  **   button or spinedit button is clicked.  If appropriate, the        **
  **   respective chart value (plotted) is also changed.                 **
  **********************************************************************}
procedure TWeb_Spinner.GenericSBDownClick(var EditBox: TEdit);
var
  Value: Integer;
begin
  Value := StrToIntDef(EditBox.Text, 0);
  if Value > 0 then Value := Value - 1;
    EditBox.Text := IntToStr(Value)
end;

procedure TWeb_Spinner.GenericSBUpClick(var EditBox: TEdit);
var
  Value: Integer;
begin
  Value := StrToIntDef(EditBox.Text, 0);
  if Value < 100 then Value := Value + 1;
    EditBox.Text := IntToStr(Value)
end;

procedure TWeb_Spinner.SpinButton1DownClick(Sender: TObject);
var
  Loop: Integer;
begin
  ChartFX6.OpenData[COD_VALUES] := MakeLong(1,10);
  ChartFX6.ThisSerie := 0;

  if Sender = SpinButton1 then GenericSBDownClick(Edit4);
  if Sender = SpinButton2 then GenericSBDownClick(Edit5);
  if Sender = SpinButton3 then GenericSBDownClick(Edit6);
  if Sender = SpinButton4 then GenericSBDownClick(Edit7);
  if Sender = SpinButton5 then GenericSBDownClick(Edit12);
  if Sender = SpinButton6 then GenericSBDownClick(Edit13);
  if Sender = SpinButton7 then GenericSBDownClick(Edit14);
  if Sender = SpinButton8 then GenericSBDownClick(Edit15);
  if Sender = SpinButton9 then GenericSBDownClick(Edit16);
  if Sender = SpinButton10 then GenericSBDownClick(Edit17);

  ChartFX6.Value[0] := StrToIntDef(Edit4.Text, 0);
  ChartFX6.Value[1] := StrToIntDef(Edit5.Text, 0);
  ChartFX6.Value[2] := StrToIntDef(Edit6.Text, 0);
```

```
   ChartFX6.Value[3]  := StrToIntDef(Edit7.Text, 0);
   ChartFX6.Value[4]  := StrToIntDef(Edit12.Text, 0);
   ChartFX6.Value[5]  := StrToIntDef(Edit13.Text, 0);
   ChartFX6.Value[6]  := StrToIntDef(Edit14.Text, 0);
   ChartFX6.Value[7]  := StrToIntDef(Edit15.Text, 0);
   ChartFX6.Value[8]  := StrToIntDef(Edit16.Text, 0);
   ChartFX6.Value[9]  := StrToIntDef(Edit17.Text, 0);

   ChartFX6.CloseData[COD_VALUES] := 0
end;

procedure TWeb_Spinner.SpinButton1UpClick(Sender: TObject);
begin
   ChartFX6.OpenData[COD_VALUES] := MakeLong(1,10);
   ChartFX6.ThisSerie := 0;

   if Sender = SpinButton1 then GenericSBUpClick(Edit4);
   if Sender = SpinButton2 then GenericSBUpClick(Edit5);
   if Sender = SpinButton3 then GenericSBUpClick(Edit6);
   if Sender = SpinButton4 then GenericSBUpClick(Edit7);
   if Sender = SpinButton5 then GenericSBUpClick(Edit12);
   if Sender = SpinButton6 then GenericSBUpClick(Edit13);
   if Sender = SpinButton7 then GenericSBUpClick(Edit14);
   if Sender = SpinButton8 then GenericSBUpClick(Edit15);
   if Sender = SpinButton9 then GenericSBUpClick(Edit16);
   if Sender = SpinButton10 then GenericSBUpClick(Edit17);

   ChartFX6.Value[0]  := StrToIntDef(Edit4.Text, 0);
   ChartFX6.Value[1]  := StrToIntDef(Edit5.Text, 0);
   ChartFX6.Value[2]  := StrToIntDef(Edit6.Text, 0);
   ChartFX6.Value[3]  := StrToIntDef(Edit7.Text, 0);
   ChartFX6.Value[4]  := StrToIntDef(Edit12.Text, 0);
   ChartFX6.Value[5]  := StrToIntDef(Edit13.Text, 0);
   ChartFX6.Value[6]  := StrToIntDef(Edit14.Text, 0);
   ChartFX6.Value[7]  := StrToIntDef(Edit15.Text, 0);
   ChartFX6.Value[8]  := StrToIntDef(Edit16.Text, 0);
   ChartFX6.Value[9]  := StrToIntDef(Edit17.Text, 0);

   ChartFX6.CloseData[COD_VALUES] := 0
end;


{*********************************************************************
 **   Resets any graph (chart) to a "risk neutral," linear curve.   **
 *********************************************************************}

procedure TWeb_Spinner.ResetGraph(var Chart: TChartFX);
var
   Loop: Integer;
begin
   Chart.OpenData[COD_VALUES] := MakeLong(1,10);
   Chart.ThisSerie := 0;

   for Loop := 0 to 9 do
     Chart.Value[Loop] := (10 - Loop) * 10;

   Chart.CloseData[COD_VALUES] := 0
```

104

```
end;


{*********************************************************************
 **   Resets the large chart on the Utility page.                  **
 ******************************************************************}

procedure TWeb_Spinner.BitBtn6Click(Sender: TObject);
begin
  ResetGraph(ChartFX6)
end;


{*********************************************************************
 **   This is the "Ok" button on the large chart panel.  It is used to  **
 **   acknowledge changes made to particular utility curves.        **

 ******************************************************************}

procedure TWeb_Spinner.BitBtn7Click(Sender: TObject);
begin
  if UtilityGraph = SetUpCost then SwitchGraphValues(UtilSetUpCostArray,
                                                   ChartFX6, ChartFX2);
  if UtilityGraph = MonthlyCost then
SwitchGraphValues(UtilMonthlyCostArray,
                                                   ChartFX6,
ChartFX4);
  if UtilityGraph = BER then SwitchGraphValues(UtilBERArray,
                                             ChartFX6, ChartFX3)
end;


{*********************************************************************
 **   This is the "Ok" button on the Basic Data tool.  When activated, **
 **   it updates the appropriate information in the appropriate global **
 **   variable.                                                    **
 ******************************************************************}

procedure TWeb_Spinner.BitBtn3Click(Sender: TObject);
begin
  LinkSpeed := StrToIntDef(ComboBox1.Text, 0);
  PacketSize := SpinEdit4.Value;
  Panel18.Visible := false
end;


{*********************************************************************
 **   Displays the current distance between nodes (distance matrix) on  **
 **   data inputs screen.                                          **
 ******************************************************************}

procedure TWeb_Spinner.BitBtn9Click(Sender: TObject);
begin
  OnExitInputBox(Sender);
  if InputPanel = Traffic then ExtractInputs(TrafficMatrix, InputMatrix)
  else if InputPanel = SetUpCost then
      ExtractInputs(SetUpCostMatrix,InputMatrix)
```

```
      else if InputPanel = MonthlyCost then ExtractInputs(MonthlyCostMatrix,
                                                          InputMatrix)
      else if InputPanel = BER then ExtractInputs(BERMatrix, InputMatrix);

    InputPanel := Distance;
    Label1.Caption := 'Distance';
    Label1.Color := clWhite;
    Panel8.Color := clWhite;
    ListBox2.Items.Clear;
    ListBox2.Items.Add('Screen Distances between ');
    ListBox2.Items.Add('nodes as drawn. (read only)');
    FillInputMatrix(DistanceMatrix, InputMatrix);
    Main(Sender)
  end;


{*********************************************************************
 ** The next two procedures provide the selection/deselection checkbox **
 ** functionality on the Solve Criteria tool.                        **
 *********************************************************************}

procedure TWeb_Spinner.CheckBox4Click(Sender: TObject);
begin
  CheckBox1.Checked := false;
  CheckBox2.Checked := false;
  CheckBox3.Checked := false;
  if NumUsedNodes > 0 then Main(Sender)
end;

procedure TWeb_Spinner.CheckBox1Click(Sender: TObject);
begin
  CheckBox4.Checked := false;
  if NumUsedNodes > 0 then Main(Sender)
end;


{*********************************************************************
 **   Identifies shortest (logical links) routes between nodes.      **
 **   Ignores endless loops if user "violates" the tree structure using **
 **   locked links.                                                  **
 *********************************************************************}

procedure TWeb_Spinner.FindLoops(Node1, Node2,
                                 Parent, SuperParent: Integer;
                              var Route: String);
var
  Loop, Count, Counter, Row, Col : Integer;
  Children: Array[1..(NumNodes - 1)] of Integer;
  Strtemp: String;
  Present: Boolean;
begin
  for loop:= 1 to (NumNodes - 1) do children[loop] := 0;

  Count := 0;
  if Parent = SuperParent then begin
    if Parent < 10 then Route := '0' + IntToStr(Parent)
    else Route := IntToStr(Parent)
```

```
end;

for Loop := 1 to NumNodes do begin
     if ((Connections[Parent, Loop] <> NoLink) and
          (Connections[Parent, Loop] <> Testing) and
          (Connections[Parent, Loop] <> Used) and
          (Connections[Parent, Loop] <> LUsed) and
          (Loop <> SuperParent)) then
     begin
       Count := Count + 1;
       Children[Count] := Loop
     end
end;

for Loop := 1 to (NumNodes - 1) do begin
   if ((Children[Loop] <> 0) and (not Found)) then begin
     if Children[Loop] = Node2 then begin
       RedundantRoutes := RedundantRoutes + 1;
       Found := true;
       if Node2 < 10 then Path := '0' + IntToStr(Node2)
       else Path := IntToStr(Node2);
       if Connections[Parent, Node2] = Linked then begin
         Connections[Parent, Node2] := Used;
         Connections[Node2, Parent] := Used
       end;
       if Connections[Parent, Node2] = LockLinked then begin
         Connections[Parent, Node2] := LUsed;
         Connections[Node2, Parent] := LUsed
       end;
     end;

     if Children[Loop] <> Node2 then begin
       Present := false; {Ensures not beginning an endless network loop}
       for Counter := 1 to (Length(Route) - 1) do begin
         if ((Route[Counter] <> ',') and (Route[Counter + 1] <> ','))
         then begin
           if Children[Loop] = (StrToInt(Route[Counter]) * 10 +
                                 StrToInt(Route[Counter + 1]))
                                 then Present := true
         end
       end;

       if ((not Present) and (not Found)) then begin
         if children[loop] < 10 then strtemp := '0'
                                       + IntToStr(children[loop])
           else strtemp := IntToStr(children[loop]);
         if Route = '' then Route := strtemp
           else Route := Route + ',' + strtemp;
         FindLoops(Node1, Node2, children[loop], Parent, Route)
       end
     end;

     if Found then begin
       if Parent < 10 then Path := Path + '-0' + IntToStr(Parent)
       else Path := Path + '-' + IntToStr(Parent);
       if Connections[Parent, SuperParent] = Linked then begin
         Connections[Parent, SuperParent] := Used;
```

```pascal
              Connections[SuperParent, Parent] := Used
            end;
            if Connections[Parent, SuperParent] = LockLinked then begin
              Connections[Parent, SuperParent] := LUsed;
              Connections[SuperParent, Parent] := LUsed
            end
          end

      end
    end
end;


{*********************************************************************
 **  Fills the Web Spinner route matrix with all logical connections.  **
 *********************************************************************}

procedure TWeb_Spinner.FillRouteMatrix;
var
  Row, Col, Row2, Col2, Node1, Node2: Integer;
  Route: String;
begin
  for Row := 1 to NumUsedNodes do begin
    for Col := 1 to NumUsedNodes do begin
      if Row > Col then begin

        Route := '';
        Path := '';
        Found := false;
        Node1 := Row;
        Node2 := Col;

        FindLoops(Node1, Node2, Node1, Node1, Route);
        RouteMatrix[Row, Col] := Path;
        RouteMatrix[Col, Row] := Path;

        for Row2 := 1 to NumUsedNodes do
          for Col2 := 1 to NumUsedNodes do begin
            if Connections[Row2, Col2] = Used
              then Connections[Row2, Col2] := Linked;
            if Connections[Row2, Col2] = LUsed
              then Connections[Row2, Col2] := LockLinked
          end

      end
    end
  end
end;


{*********************************************************************
 **  Computes Average Link Activity for physical links in the network. **
 *********************************************************************}

procedure TWeb_Spinner.ComputeAvgLinkActy;
var
  Row, Col, Loop: Integer;
```

```
   Len, Node1, Node2: Integer;
   Route: String;
begin
   for Row := 1 to NumNodes do
     for Col := 1 to NumNodes do
       AvgLinkActyMatrix[Row, Col] := 0;

   for Row := 1 to NumUsedNodes do begin
     for Col := 1 to NumUsedNodes do begin
       if Row > Col then begin

         Route := RouteMatrix[Row, Col];
         Len := Length(Route);

         for Loop := 1 to (Len - 4) do begin
           if ((Route[Loop] <> '-') and (Route[Loop + 1] <> '-')) then
           begin
             Node1 := (StrToIntDef(Route[Loop],0) * 10) +
                       StrToIntDef(Route[Loop + 1], 0);

             Node2 := (StrToIntDef(Route[Loop + 3],0) * 10) +
                       StrToIntDef(Route[Loop + 4], 0);

             if Node1 > Node2 then begin
               AvgLinkActyMatrix[Node1, Node2] :=
                         AvgLinkActyMatrix[Node1, Node2]
                         + TrafficMatrix[Row, Col];
               AvgLinkActyMatrix[Node2, Node1] :=
                         AvgLinkActyMatrix[Node1, Node2]
             end
             else begin
               AvgLinkActyMatrix[Node2, Node1] :=
                         AvgLinkActyMatrix[Node2, Node1]
                         + TrafficMatrix[Row, Col];
               AvgLinkActyMatrix[Node1, Node2] :=
                         AvgLinkActyMatrix[Node2, Node1]
             end

           end
         end
       end
     end
   end
end;


{*************************************************************************
 **   Computes and fills the Expected delay matrix for each physical   **
 **   link in the network.                                             **
 *************************************************************************}

procedure TWeb_Spinner.FillExpectedDelayMatrix;
var
   Row, Col, Loop: Integer;
   Len, Node1, Node2: Integer;
   ExpValue: Real;
   Route: String;
```

```
begin
  for Row := 1 to NumNodes do
    for Col := 1 to NumNodes do
      ExpDelayMatrix[Row, Col] := 0.0;

  for Row := 1 to NumUsedNodes do begin
    for Col := 1 to NumUsedNodes do begin
      if Row > Col then begin

        Route := RouteMatrix[Row, Col];
        Len := Length(Route);

        for Loop := 1 to (Len - 4) do begin
          if ((Route[Loop] <> '-') and (Route[Loop + 1] <> '-')) then
          begin
            Node1 := (StrToIntDef(Route[Loop],0) * 10) +
                      StrToIntDef(Route[Loop + 1], 0);

            Node2 := (StrToIntDef(Route[Loop + 3],0) * 10) +
                      StrToIntDef(Route[Loop + 4], 0);

            if AvgLinkActyMatrix[Node1, Node2] <> 0 then
              ExpValue := 1/((LinkSpeed * 1.0)/(PacketSize * 1.0)
                          - AvgLinkActyMatrix[Node1, Node2])
            else ExpValue := 0;

            ExpDelayMatrix[Row, Col] := ExpDelayMatrix[Row, Col]
                                        + ExpValue;
            ExpDelayMatrix[Col, Row] := ExpDelayMatrix[Row, Col]
          end
        end
      end
    end
  end
end;


{*********************************************************************
 **   Verifies that the user wants to exit the Web Spinner application. **
 *********************************************************************}

procedure TWeb_Spinner.Close1Click(Sender: TObject);
begin
  if MessageDlg('Exit the Web Spinner Application?', mtConfirmation,
                mbOkCancel, 0) = mrOk then begin
    About.Close;
    Web_Spinner.Close
  end
end;


{*********************************************************************
 **   This is the "Reset" button on the large chart panel on the        **
 **   Utility page.  When activated, it returns the large chart to the   **
 **   values contained when the user began modifying it.  In effect,     **
 **   it allows the user to reject recent chart modifications.           **
 *********************************************************************}
```

```
procedure TWeb_Spinner.BitBtn13Click(Sender: TObject);
begin
   if UtilityGraph = SetUpCost then SwitchGraphValues(UtilSetUpCostArray,
                                           ChartFX2, ChartFX6)
   else if UtilityGraph = BER then SwitchGraphValues(UtilBERArray,
                                           ChartFX3, ChartFX6)
   else if UtilityGraph = MonthlyCost then
       SwitchGraphValues(UtilMonthlyCostArray, ChartFX4, ChartFX6)
end;


{*********************************************************************
 **   The final routines are used to create and print the hardcopy   **
 **   report that is sent to the printer.                            **
 *********************************************************************}

procedure TWeb_Spinner.ListBox3DrawItem(Control: TWinControl;
   Index: Integer; Rect: TRect; State: TOwnerDrawState);
var
   S1, S2, S3: String;
   A: Array[0..255] of Char;
   P, P2: Integer;
begin
   P := Pos(Spacer, ListBox3.Items[Index]);
   P2 := Pos(Spacer2, ListBox3.Items[Index]);

   S1 := Copy(ListBox3.Items[Index], 1, P - 1);
   S2 := Copy(ListBox3.Items[Index], P + 1, P2 - P - 1);
   S3 := Copy(ListBox3.Items[Index], P2 + 1,
Length(ListBox3.Items[Index]));

   ListBox3.Canvas.FillRect(rect);
   DrawText(ListBox3.Canvas.Handle, StrPCopy(A, S1), -1, Rect,
            dt_SingleLine or dt_Left or dt_VCenter);

   Rect.Left := Header1.SectionWidth[0];
   DrawText(ListBox3.Canvas.Handle, StrPCopy(A, S2), -1, Rect,
            dt_SingleLine or dt_Left or dt_VCenter);

   Rect.Left := Header1.SectionWidth[0] + Header1.SectionWidth[1];
   DrawText(ListBox3.Canvas.Handle, StrPCopy(A, S3), -1, Rect,
            dt_SingleLine or dt_Left or dt_VCenter)
end;

procedure TWeb_Spinner.Header1Sizing(Sender: TObject; ASection,
   AWidth: Integer);
begin
   ListBox3.Invalidate
end;

procedure TWeb_Spinner.ListBox4DrawItem(Control: TWinControl;
   Index: Integer; Rect: TRect; State: TOwnerDrawState);
var
   S1, S2, S3: String;
   A: Array[0..255] of Char;
   P, P2: Integer;
```

```
begin
  P  := Pos(Spacer, ListBox4.Items[Index]);
  P2 := Pos(Spacer2, ListBox4.Items[Index]);

  S1 := Copy(ListBox4.Items[Index], 1, P - 1);
  S2 := Copy(ListBox4.Items[Index], P + 1, P2 - P - 1);
  S3 := Copy(ListBox4.Items[Index], P2 + 1,
             Length(ListBox4.Items[Index]));

  ListBox4.Canvas.FillRect(rect);
  DrawText(ListBox4.Canvas.Handle, StrPCopy(A, S1), -1, Rect,
           dt_SingleLine or dt_Left or dt_VCenter);

  Rect.Left := Header2.SectionWidth[0];
  DrawText(ListBox4.Canvas.Handle, StrPCopy(A, S2), -1, Rect,
           dt_SingleLine or dt_Left or dt_VCenter);

  Rect.Left := Header2.SectionWidth[0] + Header2.SectionWidth[1];
  DrawText(ListBox4.Canvas.Handle, StrPCopy(A, S3), -1, Rect,
           dt_SingleLine or dt_Left or dt_VCenter)
end;

procedure TWeb_Spinner.Header2Sizing(Sender: TObject; ASection,
  AWidth: Integer);
begin
  ListBox4.Invalidate
end;

procedure TWeb_Spinner.BitBtn14Click(Sender: TObject);
var
  Row, Col, TotalPkts, Value, Loop, Activity: Integer;
  ExpValue, TotalDelay, NetExpDelay: Real;
  Route, NewRoute : String;
begin
  FillRouteMatrix;
  ComputeAvgLinkActy;
  FillExpectedDelayMatrix;

  Edit18.Text := IntToStr(NumUsedNodes);
  Value := ComputeTotal(SetUpCostMatrix, Connections);
  Edit20.Text := '$' + IntToStr(Value);
  Value := ComputeTotal(MonthlyCostMatrix, Connections);
  Edit21.Text := '$' + IntToStr(Value);
  Edit22.Text := IntToStr(LinkSpeed) + ' bps';
  Edit23.Text := IntToStr(PacketSize) + ' bits';

  ListBox3.Items.Clear;
  ListBox4.Items.Clear;

  TotalDelay := 0.0;
  TotalPkts := 0;
  Activity := 0;

  for Row := 1 to NumUsedNodes do
    for Col := 1 to NumUsedNodes do
      if Row < Col then begin
        if Connections[Row, Col] <> NoLink then begin
```

112

```
     if AvgLinkActyMatrix[Row, Col] <> 0 then begin
        ExpValue := 1/((LinkSpeed * 1.0)/
                     (PacketSize * 1.0)
                     - AvgLinkActyMatrix[Row, Col]);

        if ((ExpValue < 0.0) and (AvgLinkActyMatrix[Row, Col]
             > Activity)) then
             Activity := AvgLinkActyMatrix[Row, Col]
     end
     else ExpValue := 0;
     TotalDelay := TotalDelay + (ExpValue
                    * AvgLinkActyMatrix[Row, Col]);
     ListBox3.Items.Add('   ' + IntToStr(Row) + '-' + IntToStr(Col)
                     + Spacer + '  ' +
                     IntToStr(AvgLinkActyMatrix[Row, Col])
                     + Spacer2 + '  ' +
                     FloatToStrF(ExpValue, ffGeneral, 3, 0)
                                 + ' sec');
  end;
  TotalPkts := TotalPkts + TrafficMatrix[Row, Col];
end;

If TotalPkts > 0 then NetExpDelay := TotalDelay / TotalPkts
else NetExpDelay := TotalDelay;
Edit19.Text := FloatToStrF(NetExpDelay, ffGeneral, 3, 0) + ' sec';

for Row := 1 to NumUsedNodes do
  for Col := 1 to NumUsedNodes do
    if Row > Col then begin
       Route := RouteMatrix[Row, Col];
       NewRoute := '';
       if Route[1] <> '0' then NewRoute := NewRoute + Route[1];

       for Loop := 2 to Length(RouteMatrix[Row, Col]) do begin
          if ((Route[Loop - 1] <> '-') and (Route[Loop] <> '0')) then
            NewRoute := NewRoute + Route[Loop];
          if ((Route[Loop - 1] = '1') and (Route[Loop] <> '-')) then
            NewRoute := NewRoute + Route[Loop - 1] + Route[Loop]
       end;

       RouteMatrix[Row, Col] := NewRoute;
       RouteMatrix[Col, Row] := RouteMatrix[Row, Col]
    end;

for Row := 1 to NumUsedNodes do
  for Col := 1 to NumUsedNodes do
    if Row < Col then
       ListBox4.Items.Add('   ' + IntToStr(Row) + '-' + IntToStr(Col)
          + Spacer + '  ' + RouteMatrix[Row, Col] + Spacer2 + '  ' +
          FloatToStrF(ExpDelayMatrix[Row, Col], ffGeneral, 3, 0)
                     + ' sec');

if Activity > 0 then begin
  MessageDlg('Current unifrom link bandwidth insufficient for
             average ' +
             'network loading.  Under current traffic pattern ' +
             'requirements, a minimim bandwidth of ' +
```

```
                    IntToStr((Activity * PacketSize) + 1) + ' is
                    required.',
                    mtInformation, [mbOk], 0);
          Panel18.Visible := true
       end
end;


{************************************************************************
 **   This is the "Print" button on the Analysis page.  It launches the **
 **   report printing process.                                          **
 ************************************************************************}

procedure TWeb_Spinner.BitBtn15Click(Sender: TObject);
var
  Len, Index, P1, P2, P3, P4: Integer;
  S1, S2, S3, S4, S5, S6: String[32];
  Items: TStringList;
begin

  Items := TStringList.Create;
  PixelsInInchx := GetDeviceCaps(Printer.Handle, LOGPIXELSX);
  TenthsOfInchPixelsY := GetDeviceCaps(Printer.Handle, LOGPIXELSY)
                           div 10;

  AmountPrinted :=  Printer.Canvas.TextHeight('X') + TenthsOfInchPixelsY;
  try
    Web_Spinner.Enabled := false;
    Printer.Title := 'Web Spinner';
    Printer.BeginDoc;
    Application.ProcessMessages;
    LineHeight := Printer.Canvas.TextHeight('X') + TenthsOfInchPixelsY;
    PrintHeader;
    PrintNetworkInfo;
    PrintColNames;

    Len := ListBox3.Items.Count;
    If ListBox4.Items.Count > Len then Len := ListBox4.Items.Count;

    for Index := 0 to (Len - 1) do begin
      Application.ProcessMessages;

      S1 := '';
      S2 := '';
      S3 := '';
      S4 := '';
      S5 := '';
      S6 := '';

      if Index < ListBox3.Items.Count then begin
        P1 := Pos(Spacer, ListBox3.Items[Index]);
        P2 := Pos(Spacer2, ListBox3.Items[Index]);

        S1 := Copy(ListBox3.Items[Index], 1, P1 - 1);
        S2 := Copy(ListBox3.Items[Index], P1 + 1, P2 - P1 - 1);
        S3 := Copy(ListBox3.Items[Index], P2 + 1,
                    Length(ListBox3.Items[Index]))
```

114

```
      end;
      if Index < ListBox4.Items.Count then begin
        P3  := Pos(Spacer, ListBox4.Items[Index]);
        P4  := Pos(Spacer2, ListBox4.Items[Index]);

        S4  := Copy(ListBox4.Items[Index], 1, P3 - 1);
        S5  := Copy(ListBox4.Items[Index], P3 + 1, P4 - P3 - 1);
        S6  := Copy(ListBox4.Items[Index], P4 + 1,
                    Length(ListBox4.Items[Index]))
      end;
      Items.AddObject('', Pointer(5));
      Items.AddObject(S1, Pointer(11));
      Items.AddObject(S2, Pointer(11));
      Items.AddObject(S3, Pointer(11));
      Items.AddObject('', Pointer(8));
      Items.AddObject(S4, Pointer(10));
      Items.AddObject(S5, Pointer(11));
      Items.AddObject(S6, Pointer(12));
      PrintLine(Items);

      if AmountPrinted + LineHeight > Printer.PageHeight then begin
        AmountPrinted := 0;
        if not Printer.Aborted then Printer.NewPage;
        PrintHeader;
        PrintColNames;
      end;
      Items.Clear
    end;

    AmountPrinted :=  AmountPrinted + (TenthsOfInchPixelsY * 4);
    if (AmountPrinted + (10 * LineHeight)) > Printer.PageHeight then
begin
      AmountPrinted := 0;
      if not Printer.Aborted then Printer.NewPage;
    end;

    PrintData;
    PrintCriteria;
    if not Printer.Aborted then Printer.EndDoc;
    Web_Spinner.Enabled := true;
  except
    on E: Exception do MessageDlg(E.Message, mtError, [mbok], 0);
  end
end;

procedure TWeb_Spinner.PrintLine(Items: TStringList);
var
  OutRect: TRect;
  Inches: Double;
  I : Integer;
begin
  OutRect.Left := 0;
  OutRect.Top := AmountPrinted;
  OutRect.Bottom := OutRect.Top + LineHeight;
  with Printer.Canvas do
    for I := 0 to Items.Count -1 do begin
      Inches := LongInt(Items.Objects[I]) * 0.1;
```

115

```
         OutRect.Right := OutRect.Left + Round(PixelsInInchx * Inches);
         if not Printer.Aborted then
            TextRect(OutRect, OutRect.Left, OutRect.Top, Items[I]);
            OutRect.Left := OutRect.Right
      end;
   AmountPrinted := AmountPrinted + (TenthsOfInchPixelsY * 2);
   if (AmountPrinted + LineHeight) > Printer.PageHeight then begin
      AmountPrinted := 0;
      if not Printer.Aborted then Printer.NewPage;
   end;
end;

procedure TWeb_Spinner.PrintHeader;
var
   SaveFont: TFont;
begin
   Printer.Canvas.Draw(5 * PixelsInInchX div 10, AmountPrinted,
Image2.Picture.Graphic);
   Printer.Canvas.Draw(Printer.PageWidth - Image2.Width -
                       (4 * PixelsInInchX div 10), AmountPrinted,
                       Image3.Picture.Graphic);
   SaveFont := TFont.Create;
   SaveFont.Assign(Printer.Canvas.Font);
   Printer.Canvas.Font.Assign(Label35.Font);
   Printer.Canvas.Font.Size := 18;
   Printer.Canvas.Font.Style := [fsBold, fsItalic, fsUnderLine];
   with Printer do begin
      if not Printer.Aborted then
              Canvas.TextOut((PageWidth div 2)
              - (Canvas.TextWidth('Network Analysis') div 2),
              AmountPrinted + LineHeight, 'Network Analysis');
      AmountPrinted := AmountPrinted + (5 * LineHeight) +
TenthsOfInchPixelsY;
   end;
   Printer.Canvas.Font.Assign(SaveFont);
   SaveFont.Free;
end;

procedure TWeb_Spinner.PrintNetworkInfo;
var
   NetInfo: TStringList;
begin
   NetInfo := TStringList.Create;
   Printer.Canvas.Font.Name := Label35.Font.Name;
   Printer.Canvas.Font.Size := 12;
   Printer.Canvas.Font.Style := [fsBold];
   AmountPrinted := AmountPrinted - (TenthsOfInchPixelsY * 2);
   with NetInfo do begin
      AddObject('', pointer(19));
      AddObject('Number of Nodes:', pointer(15));
      AddObject(Edit18.Text, pointer(6));
      AddObject('', pointer(4));
      AddObject('Expected Net Delay:', pointer(16));
      AddObject(Edit19.Text, pointer(11));
   end;
   PrintLine(NetInfo);
```

```
  NetInfo.Clear;
  with NetInfo do begin
    AddObject('', pointer(19));
    AddObject('Net Set-Up Cost:', pointer(15));
    AddObject(Edit20.Text, pointer(6));
    AddObject('', pointer(4));
    AddObject('Link Capacities:', pointer(16));
    AddObject(Edit22.Text, pointer(11));
  end;
  PrintLine(NetInfo);

  NetInfo.Clear;
  with NetInfo do begin
    AddObject('', pointer(19));
    AddObject('Monthly Net Cost:', pointer(15));
    AddObject(Edit21.Text, pointer(6));
    AddObject('', pointer(4));
    AddObject('# Bits / Packet:', pointer(16));
    AddObject(Edit23.Text, pointer(11));
  end;
  PrintLine(NetInfo);

  Printer.Canvas.Font := ListBox3.Font;
  AmountPrinted := AmountPrinted + (TenthsOfInchPixelsY * 3);
  NetInfo.free;
end;

procedure TWeb_Spinner.PrintColNames;
var
  ColNames: TStringList;
begin
  ColNames := TStringList.Create;
  Printer.Canvas.Font.Name := Label35.Font.Name;
  Printer.Canvas.Font.Size := 14;
  Printer.Canvas.Font.Style := [fsBold, fsUnderline, fsItalic];
  Printer.Canvas.Pen.Color := clBlack;
  with ColNames do begin
    AddObject('', pointer(5));
    AddObject('                        Physical Link Data
                pointer(34));
    AddObject('', pointer(7));
    AddObject('                        Logical Link Data
                pointer(33));
  end;
  PrintLine(ColNames);
  AmountPrinted := AmountPrinted + TenthsOfInchPixelsY;
  ColNames.Clear;
  Printer.Canvas.Font.Size := 11;
  Printer.Canvas.Font.Style := [fsUnderLine];
  with ColNames do begin
    AddObject('', pointer(5));
    AddObject(' Link                           ', pointer(11));
    AddObject('Avg Pkts/Sec                    ', pointer(11));
    AddObject('Mean Pkt Delay                  ', pointer(11));
    AddObject('', pointer(8));
    AddObject(' Link                           ', pointer(10));
    AddObject('Route                           ', pointer(11));
```

117

```
      AddObject('Exp Route Delay            ', pointer(12))
    end;
    PrintLine(ColNames);
    Printer.Canvas.Font := ListBox3.Font;
    AmountPrinted := AmountPrinted + TenthsOfInchPixelsY;
    ColNames.free;
end;

procedure TWeb_Spinner.PrintData;
var
   Row, Col: Integer;
   Line: TStringList;
begin
   Line := TStringList.Create;
   Printer.Canvas.Font.Name := Label35.Font.Name;
   Printer.Canvas.Font.Size := 13;
   Printer.Canvas.Font.Style := [fsBold, fsUnderline, fsItalic];
   Printer.Canvas.Pen.Color := clBlack;
   AmountPrinted := AmountPrinted + (TenthsOfInchPixelsY * 2);
   with Line do begin
     AddObject('', pointer(5));
     AddObject('Traffic Data

               pointer(41));
     AddObject('Bit Error Rates

               pointer(30));
   end;
   PrintLine(Line);
   AmountPrinted := AmountPrinted + TenthsOfInchPixelsY;

   Line.Clear;
   Printer.Canvas.Font := ListBox3.Font;
   Printer.Canvas.Font.Style := [fsUnderline];
   Printer.Canvas.Font.Size := 8;

   Line.AddObject('', pointer(5));
   for Col := 1 to NumUsedNodes do
     Line.AddObject(InttoStr(Col) + '            ', pointer (3));
   Line.AddObject('', pointer((NumNodes-NumUsedNodes) * 3 + 11));
   for Col := 1 to NumUsedNodes do
     Line.AddObject(InttoStr(Col) + '            ', pointer (3));
   PrintLine(Line);

   Line.Clear;
   Printer.Canvas.Font := ListBox3.Font;
   Printer.Canvas.Font.Size := 8;
   for Row := 1 to NumUsedNodes do begin
     Line.AddObject('', pointer(5));
     for Col := 1 to NumUsedNodes do begin
       with Line do begin
         if Row = Col then AddObject('-', pointer(3))
         else AddObject(IntToStr(TrafficMatrix[Row, Col]), pointer(3));
       end
     end;
     Line.AddObject('', pointer((NumNodes-NumUsedNodes) * 3 + 11));
     for Col := 1 to NumUsedNodes do begin
```

118

```
      with Line do begin
        if Row = Col then AddObject('-', pointer(3))
        else AddObject(IntToStr(BERMatrix[Row, Col]), pointer(3));
      end
    end;
    PrintLine(Line);
    Line.Clear
  end;
  Printer.Canvas.Font := ListBox3.Font;
  AmountPrinted := AmountPrinted + (TenthsOfInchPixelsY * 4);

  Printer.Canvas.Font.Name := Label35.Font.Name;
  Printer.Canvas.Font.Size := 13;
  Printer.Canvas.Font.Style := [fsBold, fsUnderline, fsItalic];
  Printer.Canvas.Pen.Color := clBlack;
  AmountPrinted := AmountPrinted + (TenthsOfInchPixelsY * 2);
  with Line do begin
    AddObject('', pointer(5));
    AddObject('Link Set-Up Costs
              pointer(41));
    AddObject('Monthly Link Costs
              pointer(30));
  end;
  PrintLine(Line);
  AmountPrinted := AmountPrinted + TenthsOfInchPixelsY;

  if (AmountPrinted + (12 * LineHeight)) > Printer.PageHeight then begin
    AmountPrinted := 0;
    if not Printer.Aborted then Printer.NewPage;
  end;

  Line.Clear;
  Printer.Canvas.Font := ListBox3.Font;
  Printer.Canvas.Font.Style := [fsUnderline];
  Printer.Canvas.Font.Size := 8;

  Line.AddObject('', pointer(5));
  for Col := 1 to NumUsedNodes do
    Line.AddObject(InttoStr(Col) + '            ', pointer (3));
  Line.AddObject('', pointer((NumNodes-NumUsedNodes) * 3 + 11));
  for Col := 1 to NumUsedNodes do
    Line.AddObject(InttoStr(Col) + '            ', pointer (3));
  PrintLine(Line);
  Line.Clear;

  Printer.Canvas.Font := ListBox3.Font;
  Printer.Canvas.Font.Size := 8;
  for Row := 1 to NumUsedNodes do begin
    Line.AddObject('', pointer(5));
    for Col := 1 to NumUsedNodes do begin
      with Line do begin
        if Row = Col then AddObject('-', pointer(3))
        else AddObject(IntToStr(SetUpCostMatrix[Row, Col]), pointer(3));
      end
    end;
    Line.AddObject('', pointer((NumNodes-NumUsedNodes) * 3 + 11));
    for Col := 1 to NumUsedNodes do begin
```

119

```
         with Line do begin
            if Row = Col then AddObject('-', pointer(3))
            else AddObject(IntToStr(MonthlyCostMatrix[Row, Col]),
                             pointer(3));
         end
      end;
      PrintLine(Line);
      Line.Clear
   end;
   Printer.Canvas.Font := ListBox3.Font;
   AmountPrinted := AmountPrinted + TenthsOfInchPixelsY;
   Line.free
end;

procedure TWeb_Spinner.PrintCriteria;
var
   Line: TStringList;
   Criteria, Criteria2, Criteria3: String;
begin
   Line := TStringList.Create;
   Printer.Canvas.Font.Assign(Label35.Font);
   Printer.Canvas.Font.Size := 11;
   Printer.Canvas.Font.Style := [];
   AmountPrinted := AmountPrinted + (TenthsOfInchPixelsY * 4);
   Criteria := '';
   Criteria2 := '';
   Criteria3 := '';
   if CheckBox4.Checked then Criteria := 'Distance'
   else begin
      if CheckBox1.Checked then Criteria := 'SetUp Cost
      if CheckBox2.Checked then Criteria2 := 'Monthly Cost
      if CheckBox3.Checked then Criteria3 := 'BER'
   end;
   Printer.Canvas.Font.Style := [fsBold, fsUnderline];
   Line.Clear;
   with Line do begin
      AddObject('', pointer(5));
      AddObject('Criteria Used for Network Generation', pointer(50))
   end;
   PrintLine(Line);
   Printer.Canvas.Font.Style := [fsItalic];

   Line.Clear;
   with Line do begin
      AddObject('', pointer(5));
      AddObject(Criteria, pointer(11));
      AddObject('', pointer(2));
      AddObject(Criteria2, pointer(11));
      AddObject('', pointer(2));
      AddObject(Criteria3, pointer(11));
   end;
   PrintLine(Line);
   Printer.Canvas.Font.Style := [];

   AmountPrinted := AmountPrinted + (TenthsOfInchPixelsY * 2);
   Line.Clear;
   with Line do begin
```

```
      AddObject('', pointer(5));
      AddObject('Set-Up Cost Weight    =', pointer(20));
      AddObject(IntToStr(SpinEdit1.Value) + '%', pointer(10));
   end;
   PrintLine(Line);

   Line.Clear;
   with Line do begin
      AddObject('', pointer(5));
      AddObject('Monthly Cost Weight =', pointer(20));
      AddObject(IntToStr(SpinEdit3.Value) + '%', pointer(10));
   end;
   PrintLine(Line);

   Line.Clear;
   with Line do begin
      AddObject('', pointer(5));
      AddObject('BER Cost Weight       =', pointer(20));
      AddObject(IntToStr(SpinEdit2.Value) + '%', pointer(10));
   end;
   PrintLine(Line);

   Printer.Canvas.Font.Assign(ListBox3.Font);
   Line.Free
end;

end.
```

LIST OF REFERENCES

1. Sridhar, Suresh, "Evaluating Network Performance," Class Notes -- Advanced Topics on Networks, 1995.

2. Kleinrock L., *Queuing Systems*, vol. I, John Wiley & Sons, Inc., 1975.

BIBLIOGRAPHY

Boldon, B., Deo, N., and Kumar, N., *Minimum-Weight Degree-Constrained Spanning Tree Problem: Heuristics and Implementation on an SIMD Parallel Machine*, Tech. Rpt. CS-TR-95-02, Department of Computer Science, University of Central Florida, Orlando, FL, 1995.

Camerini, P. M., Fratta, L., and Maffioli, F., *Some Results on the Design of Tree-Structured Communication Networks*, Istituto di Electtronica and Centro di Telecomunicazioni Spaziali of CMR, Milano, Italy 1979.

Cantú, M., *Mastering Delphi*, SYBEX, Inc., 1995.

Dean, N., Mevenkamp, M., and Monma, C. L., "NETPAD: An Interactive Graphics System for Network Modeling and Optimization," *Operations Research Letters*, vol. 17, no. 2, pp.89-101, Institute for Operations Research and the Management Sciences (INFORMS), Morristown, NJ, March 1995.

Evans, J. R. and Minieka, E., *Optimization Algorithms for Networks and Graphs*, 2nd ed., Marcel Dekker, Inc., 1992.

Fitzgerald, J., *Business Data Communications*, 4th ed., John Wiley & Sons, Inc., 1993.

Frank, H. and Frisch, I. T., *Communication, Transmission, and Transportation Networks*, Addison-Wesley Publishing Company, Inc., 1971.

Miller, G. M., *Modern Electronic Communication*, 4th ed., Prentice-Hall, Inc., 1993.

Object Pascal Language Guide, ver. 1.0., Borland International, Inc., 1995.

Pacheco, X. and Teixeira, S., *Delphi® Developer's Guide*, 1st ed., Borland Press and Sams Publishing., Inc., 1995.

Phillips, D. T. and Garcia-Diaz, A., *Fundamentals of Network Analysis*, Prentice-Hall, Inc., 1981.

Taylor, B. W., III, *Introduction to Management Science*, 4th ed., Allyn and Bacon, Inc., 1993.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road., Ste 0944
   Fort Belvoir, VA, 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Dr. Suresh Sridhar, Code SM/SS 1
   Naval Postgraduate School
   Monterey, California 93943

4. Dr. Hemant Bhargava, Code SM/BH 1
   Naval Postgraduate School
   Monterey, California 93943

5. Jeffrey A, Margraf, LT, USN 2
   1491 W. Co. Rd. 11
   Tiffin, Ohio   44883